IN THE UNITED STATES DISTRICT COURT
FOR THE EASTERN DISTRICT OF TEXAS
MARSHALL DIVISION

IMPLICIT, LLC,	§	
	§	
Plaintiff,	§	
	§	**Civil Action No. 2:19-cv-37-JRG-RSP**
v.	§	
	§	**JURY TRIAL DEMANDED**
JUNIPER NETWORKS, INC.,	§	
	§	
Defendant.	§	

FIRST AMENDED COMPLAINT FOR PATENT INFRINGEMENT

COMES NOW Plaintiff Implicit, LLC ("Implicit") and files this First Amended Complaint for Patent Infringement against Defendant Juniper Networks, Inc. ("Juniper"), alleging as follows:

I. NATURE OF THE SUIT

1. This is a claim for patent infringement arising under the patent laws of the United States, Title 35 of the United States Code.

II. THE PARTIES

2. Plaintiff **Implicit, LLC** is a Washington limited liability company that maintains its principal place of business in Tyler, Texas.

3. Defendant **Juniper Networks, Inc.** is a Delaware corporation that does business in Texas, directly or through intermediaries, maintains a principal place of business in Sunnyvale, California, and maintains a regular and established place of business in Plano, Texas.

III. JURISDICTION AND VENUE

4. This action arises under the patent laws of the United States, Title 35 of the United States Code. Thus, this Court has subject matter jurisdiction pursuant to 28 U.S.C. §§ 1331 and 1338(a).

5. This Court has general personal jurisdiction over Juniper because Juniper maintains a regular and established place of business in Plano, Texas.

6. Further, this Court has specific personal jurisdiction over Juniper in this action pursuant to due process and the Texas Long Arm Statute because the claims asserted herein arise out of or are related to Juniper's voluntary contacts with this forum, such voluntary contacts including but not limited to: (i) at least a portion of the actions complained of herein; (ii) purposefully and voluntarily placing one or more Accused Products into the stream of commerce with the expectation that they will be purchased by consumers in this forum; or (iii) regularly doing or soliciting business, engaging in other persistent courses of conduct, or deriving substantial revenue from Accused Products provided to individuals in Texas and in this District.

7. Venue is proper in this Court under 28 U.S.C. §§ 1391(b)(3) and 1400(b) for at least the reasons set forth above and because Juniper maintains a regular and established place of business in Plano, Texas, which is in this District.

IV. BACKGROUND

A. The Asserted Patents

8. This cause of action asserts infringement of United States Patent Nos. 8,056,075 (the "'075 Patent"); 8,856,779 (the "'779 Patent"); 9,325,740 (the "'740 Patent"); 8,694,683 (the "'683 Patent"); 9,270,790 (the "'790 Patent"); 9,591,104 (the "'104 Patent"); 10,027,780 (the

"'780 Patent"); 10,033,839 (the "'839 Patent"); and 10,225,378 (the "'378 Patent") (collectively, the "Asserted Patents").

9. A true and correct copy of the '075 Patent, entitled "Server Request Management," and with Edward Balassanian as the named inventor, is attached hereto as Exhibit 1.

10. The '075 Patent duly and legally issued on November 8, 2011.

11. Implicit is the current owner by assignment of all rights, title, and interest in and under the '075 Patent. Implicit has standing to sue for infringement of the '075 Patent.

12. A true and correct copy of the '779 Patent, entitled "Application Server for Delivering Applets to Client Computing Devices in a Distributed Environment," and with Edward Balassanian as the named inventor, is attached hereto as Exhibit 2.

13. The '779 Patent duly and legally issued on October 7, 2014.

14. Implicit is the current owner by assignment of all rights, title, and interest in and under the '779 Patent. Implicit has standing to sue for infringement of the '779 Patent.

15. A true and correct copy of the '740 Patent, entitled "Application Server for Delivering Applets to Client Computing Devices in a Distributed Environment," and with Edward Balassanian as the named inventor, is attached hereto as Exhibit 3.

16. The '740 Patent duly and legally issued on April 26, 2016.

17. Implicit is the current owner by assignment of all rights, title, and interest in and under the '740 Patent. Implicit has standing to sue for infringement of the '740 Patent.

18. A true and correct copy of the '683 Patent, entitled "Method and System for Data Demultiplexing," and with Edward Balassanian as the named inventor, is attached hereto as Exhibit 4.

19. The '683 Patent duly and legally issued on April 8, 2014.

20. Implicit is the current owner by assignment of all rights, title, and interest in and under the ’683 Patent. Implicit has standing to sue for infringement of the ’683 Patent.

21. A true and correct copy of the ’790 Patent, entitled “Method and System for Data Demultiplexing,” and with Edward Balassanian as the named inventor, is attached hereto as Exhibit 5.

22. The ’790 Patent duly and legally issued on February 23, 2016.

23. Implicit is the current owner by assignment of all rights, title, and interest in and under the ’790 Patent. Implicit has standing to sue for infringement of the ’790 Patent.

24. A true and correct copy of the ’104 Patent, entitled “Method and System for Data Demultiplexing,” and with Edward Balassanian as the named inventor, is attached hereto as Exhibit 6.

25. The ’104 Patent duly and legally issued on March 7, 2017.

26. Implicit is the current owner by assignment of all rights, title, and interest in and under the ’104 Patent. Implicit has standing to sue for infringement of the ’104 Patent.

27. A true and correct copy of the ’780 Patent, entitled “Method and System for Data Demultiplexing,” and with Edward Balassanian as the named inventor, is attached hereto as Exhibit 7.

28. The ’780 Patent duly and legally issued on July 17, 2018.

29. Implicit is the current owner by assignment of all rights, title, and interest in and under the ’780 Patent. Implicit has standing to sue for infringement of the ’780 Patent.

30. A true and correct copy of the ’839 Patent, entitled “Method and System for Data Demultiplexing,” and with Edward Balassanian as the named inventor, is attached hereto as Exhibit 8.

31. The ’839 Patent duly and legally issued on July 24, 2018.

32. Implicit is the current owner by assignment of all rights, title, and interest in and under the ’839 Patent. Implicit has standing to sue for infringement of the ’839 Patent.

33. A true and correct copy of the ’378 Patent, entitled “Method and System for Data Demultiplexing,” and with Edward Balassanian as the named inventor, is attached hereto as Exhibit 9.

34. The ’378 Patent duly and legally issued on March 5, 2019.

35. Implicit is the current owner by assignment of all rights, title, and interest in and under the ’378 Patent. Implicit has standing to sue for infringement of the ’378 Patent.

B. Juniper

36. Juniper, directly or through intermediaries, makes, uses, sells, or offers to sell within the United States, or imports into the United States, certain products (the “Accused Products”), including but not limited to Juniper SRX Series Services Gateways.

37. By selling or offering to sell the Accused Products, Juniper, directly or through intermediaries, purposefully and voluntarily places the Accused Products into the stream of commerce with the expectation that they will be purchased or used by consumers in this District.

V. NOTICE

38. The allegations of each foregoing paragraph are incorporated by reference as if fully set forth herein.

39. At least by filing and serving the Original Complaint in this action, Implicit has given Juniper written notice of the ’075 Patent, the ’779 Patent, the ’740 Patent, the ’683 Patent, the ’790 Patent, the ’104 Patent, the ’780 Patent, and the ’839 Patent and of Juniper’s infringement thereof.

40. At least by filing and serving this First Amended Complaint, Implicit has given Juniper written notice of the ’378 Patent and of Juniper’s infringement thereof.

VI. CLAIMS

A. Infringement of the ’075 Patent

41. The allegations of each foregoing paragraph are incorporated by reference as if fully set forth herein and form the basis for the following cause of action against Juniper.

42. The Accused Products are covered by at least claim 1 of the ’075 Patent.

43. Juniper has directly infringed and continues to infringe at least claim 1 of the ’075 Patent in violation of 35 U.S.C. § 271(a) by, directly or through intermediaries and without Implicit’s authority, making, using, selling, or offering to sell the Accused Products in the United States, or importing the Accused Products into the United States.

44. Further and in the alternative, at least since the filing and service of the Original Complaint in this action, Juniper has been and now is actively inducing infringement of at least claim 1 of the ’075 Patent in violation of 35 U.S.C. § 271(b). Users of the Accused Products directly infringe at least claim 1 of the ’075 Patent when they use the Accused Products in the ordinary, customary, and intended way. Juniper’s inducements include, without limitation and with specific intent to encourage the infringement, knowingly inducing consumers to use the Accused Products within the United States in the ordinary, customary, and intended way by, directly or through intermediaries, supplying the Accused Products to consumers within the United States and instructing such consumers (for example in instructional manuals or videos that Juniper provides online or with the Accused Products) how to use the Accused Products in the ordinary, customary, and intended way, which Juniper knows or should know infringes at least claim 1 of the ’075 Patent. Juniper’s inducements may further include, without limitation and with specific

intent to encourage the infringement, knowingly inducing manufacturers to make the Accused Products within the United States, or knowingly inducing distributors or resellers to sell or offer to sell the Accused Products within the United States, by, directly or through intermediaries, instructing such manufacturers, distributors, or resellers to make, sell, or offer to sell the Accused Products in the United States, which Juniper knows or should know infringes at least claim 1 of the ’075 Patent.

45. Further and in the alternative, at least since the filing and service of the Original Complaint in this action, Juniper has been and now is actively contributing to infringement of at least claim 1 of the ’075 Patent in violation of 35 U.S.C. § 271(c). Juniper installs, configures, and sells the Accused Products with one or more distinct components, including components that implement reverse-web proxy functionality (collectively, the “Server Accused Components”), each of which is especially made or especially adapted to practice the invention claimed in at least claim 1 of the ’075 Patent. Each Server Accused Component within the Accused Products constitutes a material part of the claimed invention recited in at least claim 1 of the ’075 Patent and not a staple article or commodity of commerce because it is specifically configured according to at least claim 1 of the ’075 Patent. Juniper’s contributions include, without limitation, making, offering to sell, and/or selling within the United States, and/or importing into the United States, the Accused Products, which include one or more Server Accused Components, knowing each Server Accused Component to be especially made or especially adapted for use in an infringement of at least claim 1 of the ’075 Patent, and not a staple article or commodity of commerce suitable for substantial noninfringing use.

46. As of the filing and service of the Original Complaint in this action, Juniper’s infringement of the ’075 Patent has been and continues to be willful and deliberate.

B. Infringement of the ’779 Patent

47. The allegations of each foregoing paragraph are incorporated by reference as if fully set forth herein and form the basis for the following cause of action against Juniper.

48. The Accused Products are covered by at least claim 1 of the ’779 Patent.

49. Juniper has directly infringed and continues to infringe at least claim 1 of the ’779 Patent in violation of 35 U.S.C. § 271(a) by, directly or through intermediaries and without Implicit’s authority, making, using, selling, or offering to sell the Accused Products in the United States, or importing the Accused Products into the United States.

50. Further and in the alternative, at least since the filing and service of the Original Complaint in this action, Juniper has been and now is actively inducing infringement of at least claim 1 of the ’779 Patent in violation of 35 U.S.C. § 271(b). Users of the Accused Products directly infringe at least claim 1 of the ’779 Patent when they use the Accused Products in the ordinary, customary, and intended way. Juniper’s inducements include, without limitation and with specific intent to encourage the infringement, knowingly inducing consumers to use the Accused Products within the United States in the ordinary, customary, and intended way by, directly or through intermediaries, supplying the Accused Products to consumers within the United States and instructing such consumers (for example in instructional manuals or videos that Juniper provides online or with the Accused Products) how to use the Accused Products in the ordinary, customary, and intended way, which Juniper knows or should know infringes at least claim 1 of the ’779 Patent. Juniper’s inducements may further include, without limitation and with specific intent to encourage the infringement, knowingly inducing manufacturers to make the Accused Products within the United States, or knowingly inducing distributors or resellers to sell or offer to sell the Accused Products within the United States, by, directly or through intermediaries,

instructing such manufacturers, distributors, or resellers to make, sell, or offer to sell the Accused Products in the United States, which Juniper knows or should know infringes at least claim 1 of the ’779 Patent.

51. Further and in the alternative, at least since the filing and service of the Original Complaint in this action, Juniper has been and now is actively contributing to infringement of at least claim 1 of the ’779 Patent in violation of 35 U.S.C. § 271(c). Juniper installs, configures, and sells the Accused Products with the Server Accused Components, each of which is especially made or especially adapted to practice the invention claimed in at least claim 1 of the ’779 Patent. Each Server Accused Component within the Accused Products constitutes a material part of the claimed invention recited in at least claim 1 of the ’779 Patent and not a staple article or commodity of commerce because it is specifically configured according to at least claim 1 of the ’779 Patent. Juniper’s contributions include, without limitation, making, offering to sell, and/or selling within the United States, and/or importing into the United States, the Accused Products, which include one or more Server Accused Components, knowing each Server Accused Component to be especially made or especially adapted for use in an infringement of at least claim 1 of the ’779 Patent, and not a staple article or commodity of commerce suitable for substantial noninfringing use.

52. As of the filing and service of the Original Complaint in this action, Juniper’s infringement of the ’779 Patent has been and continues to be willful and deliberate.

C. Infringement of the ’740 Patent

53. The allegations of each foregoing paragraph are incorporated by reference as if fully set forth herein and form the basis for the following cause of action against Juniper.

54. The Accused Products are covered by at least claim 1 of the ’740 Patent.

55. Juniper has directly infringed and continues to infringe at least claim 1 of the ’740 Patent in violation of 35 U.S.C. § 271(a) by, directly or through intermediaries and without Implicit’s authority, making, using, selling, or offering to sell the Accused Products in the United States, or importing the Accused Products into the United States.

56. Further and in the alternative, at least since the filing and service of the Original Complaint in this action, Juniper has been and now is actively inducing infringement of at least claim 1 of the ’740 Patent in violation of 35 U.S.C. § 271(b). Users of the Accused Products directly infringe at least claim 1 of the ’740 Patent when they use the Accused Products in the ordinary, customary, and intended way. Juniper’s inducements include, without limitation and with specific intent to encourage the infringement, knowingly inducing consumers to use the Accused Products within the United States in the ordinary, customary, and intended way by, directly or through intermediaries, supplying the Accused Products to consumers within the United States and instructing such consumers (for example in instructional manuals or videos that Juniper provides online or with the Accused Products) how to use the Accused Products in the ordinary, customary, and intended way, which Juniper knows or should know infringes at least claim 1 of the ’740 Patent. Juniper’s inducements may further include, without limitation and with specific intent to encourage the infringement, knowingly inducing manufacturers to make the Accused Products within the United States, or knowingly inducing distributors or resellers to sell or offer to sell the Accused Products within the United States, by, directly or through intermediaries, instructing such manufacturers, distributors, or resellers to make, sell, or offer to sell the Accused Products in the United States, which Juniper knows or should know infringes at least claim 1 of the ’740 Patent.

57. Further and in the alternative, at least since the filing and service of the Original Complaint in this action, Juniper has been and now is actively contributing to infringement of at least claim 1 of the ’740 Patent in violation of 35 U.S.C. § 271(c). Juniper installs, configures, and sells the Accused Products with the Server Accused Components, each of which is especially made or especially adapted to practice the invention claimed in at least claim 1 of the ’740 Patent. Each Server Accused Component within the Accused Products constitutes a material part of the claimed invention recited in at least claim 1 of the ’740 Patent and not a staple article or commodity of commerce because it is specifically configured according to at least claim 1 of the ’740 Patent. Juniper’s contributions include, without limitation, making, offering to sell, and/or selling within the United States, and/or importing into the United States, the Accused Products, which include one or more Server Accused Components, knowing each Server Accused Component to be especially made or especially adapted for use in an infringement of at least claim 1 of the ’740 Patent, and not a staple article or commodity of commerce suitable for substantial noninfringing use.

58. As of the filing and service of the Original Complaint in this action, Juniper’s infringement of the ’740 Patent has been and continues to be willful and deliberate.

D. Infringement of the ’683 Patent

59. The allegations of each foregoing paragraph are incorporated by reference as if fully set forth herein and form the basis for the following cause of action against Juniper.

60. The Accused Products are covered by at least claim 1 of the ’683 Patent.

61. Juniper has directly infringed and continues to infringe at least claim 1 of the ’683 Patent in violation of 35 U.S.C. § 271(a) by, directly or through intermediaries and without

Implicit's authority, making, using, selling, or offering to sell the Accused Products in the United States, or importing the Accused Products into the United States.

62. Further and in the alternative, at least since the filing and service of the Original Complaint in this action, Juniper has been and now is actively inducing infringement of at least claim 1 of the '683 Patent in violation of 35 U.S.C. § 271(b). Users of the Accused Products directly infringe at least claim 1 of the '683 Patent when they use the Accused Products in the ordinary, customary, and intended way. Juniper's inducements include, without limitation and with specific intent to encourage the infringement, knowingly inducing consumers to use the Accused Products within the United States in the ordinary, customary, and intended way by, directly or through intermediaries, supplying the Accused Products to consumers within the United States and instructing such consumers (for example in instructional manuals or videos that Juniper provides online or with the Accused Products) how to use the Accused Products in the ordinary, customary, and intended way, which Juniper knows or should know infringes at least claim 1 of the '683 Patent. Juniper's inducements may further include, without limitation and with specific intent to encourage the infringement, knowingly inducing manufacturers to make the Accused Products within the United States, or knowingly inducing distributors or resellers to sell or offer to sell the Accused Products within the United States, by, directly or through intermediaries, instructing such manufacturers, distributors, or resellers to make, sell, or offer to sell the Accused Products in the United States, which Juniper knows or should know infringes at least claim 1 of the '683 Patent.

63. Further and in the alternative, at least since the filing and service of the Original Complaint in this action, Juniper has been and now is actively contributing to infringement of at least claim 1 of the '683 Patent in violation of 35 U.S.C. § 271(c). Juniper installs, configures,

and sells the Accused Products with one or more distinct components, including components that implement flow-based processing and the ability to inspect application data on TCP traffic (collectively, the "Demux Accused Components"), each of which is especially made or especially adapted to practice the invention claimed in at least claim 1 of the '683 Patent. Each Demux Accused Component within the Accused Products constitutes a material part of the claimed invention recited in at least claim 1 of the '683 Patent and not a staple article or commodity of commerce because it is specifically configured according to at least claim 1 of the '683 Patent. Juniper's contributions include, without limitation, making, offering to sell, and/or selling within the United States, and/or importing into the United States, the Accused Products, which include one or more Demux Accused Components, knowing each Demux Accused Component to be especially made or especially adapted for use in an infringement of at least claim 1 of the '683 Patent, and not a staple article or commodity of commerce suitable for substantial noninfringing use.

64. As of the filing and service of the Original Complaint in this action, Juniper's infringement of the '683 Patent has been and continues to be willful and deliberate.

E. Infringement of the '790 Patent

65. The allegations of each foregoing paragraph are incorporated by reference as if fully set forth herein and form the basis for the following cause of action against Juniper.

66. The Accused Products are covered by at least claim 1 of the '790 Patent.

67. Juniper has directly infringed and continues to infringe at least claim 1 of the '790 Patent in violation of 35 U.S.C. § 271(a) by, directly or through intermediaries and without Implicit's authority, making, using, selling, or offering to sell the Accused Products in the United States, or importing the Accused Products into the United States.

68. Further and in the alternative, at least since the filing and service of the Original Complaint in this action, Juniper has been and now is actively inducing infringement of at least claim 1 of the ’790 Patent in violation of 35 U.S.C. § 271(b). Users of the Accused Products directly infringe at least claim 1 of the ’790 Patent when they use the Accused Products in the ordinary, customary, and intended way. Juniper’s inducements include, without limitation and with specific intent to encourage the infringement, knowingly inducing consumers to use the Accused Products within the United States in the ordinary, customary, and intended way by, directly or through intermediaries, supplying the Accused Products to consumers within the United States and instructing such consumers (for example in instructional manuals or videos that Juniper provides online or with the Accused Products) how to use the Accused Products in the ordinary, customary, and intended way, which Juniper knows or should know infringes at least claim 1 of the ’790 Patent. Juniper’s inducements may further include, without limitation and with specific intent to encourage the infringement, knowingly inducing manufacturers to make the Accused Products within the United States, or knowingly inducing distributors or resellers to sell or offer to sell the Accused Products within the United States, by, directly or through intermediaries, instructing such manufacturers, distributors, or resellers to make, sell, or offer to sell the Accused Products in the United States, which Juniper knows or should know infringes at least claim 1 of the ’790 Patent.

69. Further and in the alternative, at least since the filing and service of the Original Complaint in this action, Juniper has been and now is actively contributing to infringement of at least claim 1 of the ’790 Patent in violation of 35 U.S.C. § 271(c). Juniper installs, configures, and sells the Accused Products with the Demux Accused Components, each of which is especially made or especially adapted to practice the invention claimed in at least claim 1 of the ’790 Patent.

Each Demux Accused Component within the Accused Products constitutes a material part of the claimed invention recited in at least claim 1 of the ’790 Patent and not a staple article or commodity of commerce because it is specifically configured according to at least claim 1 of the ’790 Patent. Juniper’s contributions include, without limitation, making, offering to sell, and/or selling within the United States, and/or importing into the United States, the Accused Products, which include one or more Demux Accused Components, knowing each Demux Accused Component to be especially made or especially adapted for use in an infringement of at least claim 1 of the ’790 Patent, and not a staple article or commodity of commerce suitable for substantial noninfringing use.

70. As of the filing and service of the Original Complaint in this action, Juniper’s infringement of the ’790 Patent has been and continues to be willful and deliberate.

F. Infringement of the ’104 Patent

71. The allegations of each foregoing paragraph are incorporated by reference as if fully set forth herein and form the basis for the following cause of action against Juniper.

72. The Accused Products are covered by at least claim 1 of the ’104 Patent.

73. Juniper has directly infringed and continues to infringe at least claim 1 of the ’104 Patent in violation of 35 U.S.C. § 271(a) by, directly or through intermediaries and without Implicit’s authority, making, using, selling, or offering to sell the Accused Products in the United States, or importing the Accused Products into the United States.

74. Further and in the alternative, at least since the filing and service of the Original Complaint in this action, Juniper has been and now is actively inducing infringement of at least claim 1 of the ’104 Patent in violation of 35 U.S.C. § 271(b). Users of the Accused Products directly infringe at least claim 1 of the ’104 Patent when they use the Accused Products in the

ordinary, customary, and intended way. Juniper's inducements include, without limitation and with specific intent to encourage the infringement, knowingly inducing consumers to use the Accused Products within the United States in the ordinary, customary, and intended way by, directly or through intermediaries, supplying the Accused Products to consumers within the United States and instructing such consumers (for example in instructional manuals or videos that Juniper provides online or with the Accused Products) how to use the Accused Products in the ordinary, customary, and intended way, which Juniper knows or should know infringes at least claim 1 of the '104 Patent. Juniper's inducements may further include, without limitation and with specific intent to encourage the infringement, knowingly inducing manufacturers to make the Accused Products within the United States, or knowingly inducing distributors or resellers to sell or offer to sell the Accused Products within the United States, by, directly or through intermediaries, instructing such manufacturers, distributors, or resellers to make, sell, or offer to sell the Accused Products in the United States, which Juniper knows or should know infringes at least claim 1 of the '104 Patent.

75. Further and in the alternative, at least since the filing and service of the Original Complaint in this action, Juniper has been and now is actively contributing to infringement of at least claim 1 of the '104 Patent in violation of 35 U.S.C. § 271(c). Juniper installs, configures, and sells the Accused Products with the Demux Accused Components, each of which is especially made or especially adapted to practice the invention claimed in at least claim 1 of the '104 Patent. Each Demux Accused Component within the Accused Products constitutes a material part of the claimed invention recited in at least claim 1 of the '104 Patent and not a staple article or commodity of commerce because it is specifically configured according to at least claim 1 of the '104 Patent. Juniper's contributions include, without limitation, making, offering to sell, and/or selling within

the United States, and/or importing into the United States, the Accused Products, which include one or more Demux Accused Components, knowing each Demux Accused Component to be especially made or especially adapted for use in an infringement of at least claim 1 of the ’104 Patent, and not a staple article or commodity of commerce suitable for substantial noninfringing use.

76. As of the filing and service of the Original Complaint in this action, Juniper’s infringement of the ’104 Patent has been and continues to be willful and deliberate.

G. Infringement of the ’780 Patent

77. The allegations of each foregoing paragraph are incorporated by reference as if fully set forth herein and form the basis for the following cause of action against Juniper.

78. The Accused Products are covered by at least claim 1 of the ’780 Patent.

79. Juniper has directly infringed and continues to infringe at least claim 1 of the ’780 Patent in violation of 35 U.S.C. § 271(a) by, directly or through intermediaries and without Implicit’s authority, making, using, selling, or offering to sell the Accused Products in the United States, or importing the Accused Products into the United States.

80. Further and in the alternative, at least since the filing and service of the Original Complaint in this action, Juniper has been and now is actively inducing infringement of at least claim 1 of the ’780 Patent in violation of 35 U.S.C. § 271(b). Users of the Accused Products directly infringe at least claim 1 of the ’780 Patent when they use the Accused Products in the ordinary, customary, and intended way. Juniper’s inducements include, without limitation and with specific intent to encourage the infringement, knowingly inducing consumers to use the Accused Products within the United States in the ordinary, customary, and intended way by, directly or through intermediaries, supplying the Accused Products to consumers within the United

States and instructing such consumers (for example in instructional manuals or videos that Juniper provides online or with the Accused Products) how to use the Accused Products in the ordinary, customary, and intended way, which Juniper knows or should know infringes at least claim 1 of the ’780 Patent. Juniper’s inducements may further include, without limitation and with specific intent to encourage the infringement, knowingly inducing manufacturers to make the Accused Products within the United States, or knowingly inducing distributors or resellers to sell or offer to sell the Accused Products within the United States, by, directly or through intermediaries, instructing such manufacturers, distributors, or resellers to make, sell, or offer to sell the Accused Products in the United States, which Juniper knows or should know infringes at least claim 1 of the ’780 Patent.

81. Further and in the alternative, at least since the filing and service of the Original Complaint in this action, Juniper has been and now is actively contributing to infringement of at least claim 1 of the ’780 Patent in violation of 35 U.S.C. § 271(c). Juniper installs, configures, and sells the Accused Products with the Demux Accused Components, each of which is especially made or especially adapted to practice the invention claimed in at least claim 1 of the ’780 Patent. Each Demux Accused Component within the Accused Products constitutes a material part of the claimed invention recited in at least claim 1 of the ’780 Patent and not a staple article or commodity of commerce because it is specifically configured according to at least claim 1 of the ’780 Patent. Juniper’s contributions include, without limitation, making, offering to sell, and/or selling within the United States, and/or importing into the United States, the Accused Products, which include one or more Demux Accused Components, knowing each Demux Accused Component to be especially made or especially adapted for use in an infringement of at least claim 1 of the ’780

Patent, and not a staple article or commodity of commerce suitable for substantial noninfringing use.

82. As of the filing and service of the Original Complaint in this action, Juniper's infringement of the '780 Patent has been and continues to be willful and deliberate.

H. Infringement of the '839 Patent

83. The allegations of each foregoing paragraph are incorporated by reference as if fully set forth herein and form the basis for the following cause of action against Juniper.

84. The Accused Products are covered by claim 1 of the '839 Patent.

85. Juniper has directly infringed and continues to infringe claim 1 of the '839 Patent in violation of 35 U.S.C. § 271(a) by, directly or through intermediaries and without Implicit's authority, making, using, selling, or offering to sell the Accused Products in the United States, or importing the Accused Products into the United States.

86. Further and in the alternative, at least since the filing and service of the Original Complaint in this action, Juniper has been and now is actively inducing infringement of claim 1 of the '839 Patent in violation of 35 U.S.C. § 271(b). Users of the Accused Products directly infringe claim 1 of the '839 Patent when they use the Accused Products in the ordinary, customary, and intended way. Juniper's inducements include, without limitation and with specific intent to encourage the infringement, knowingly inducing consumers to use the Accused Products within the United States in the ordinary, customary, and intended way by, directly or through intermediaries, supplying the Accused Products to consumers within the United States and instructing such consumers (for example in instructional manuals or videos that Juniper provides online or with the Accused Products) how to use the Accused Products in the ordinary, customary, and intended way, which Juniper knows or should know infringes claim 1 of the '839 Patent.

Juniper's inducements may further include, without limitation and with specific intent to encourage the infringement, knowingly inducing manufacturers to make the Accused Products within the United States, or knowingly inducing distributors or resellers to sell or offer to sell the Accused Products within the United States, by, directly or through intermediaries, instructing such manufacturers, distributors, or resellers to make, sell, or offer to sell the Accused Products in the United States, which Juniper knows or should know infringes claim 1 of the '839 Patent.

87. Further and in the alternative, at least since the filing and service of the Original Complaint in this action, Juniper has been and now is actively contributing to infringement of claim 1 of the '839 Patent in violation of 35 U.S.C. § 271(c). Juniper installs, configures, and sells the Accused Products with the Demux Accused Components, each of which is especially made or especially adapted to practice the invention claimed in claim 1 of the '839 Patent. Each Demux Accused Component within the Accused Products constitutes a material part of the claimed invention recited in claim 1 of the '839 Patent and not a staple article or commodity of commerce because it is specifically configured according to claim 1 of the '839 Patent. Juniper's contributions include, without limitation, making, offering to sell, and/or selling within the United States, and/or importing into the United States, the Accused Products, which include one or more Demux Accused Components, knowing each Demux Accused Component to be especially made or especially adapted for use in an infringement of claim 1 of the '839 Patent, and not a staple article or commodity of commerce suitable for substantial noninfringing use.

88. As of the filing and service of the Original Complaint in this action, Juniper's infringement of the '839 Patent has been and continues to be willful and deliberate.

I. Infringement of the '378 Patent

89. The allegations of each foregoing paragraph are incorporated by reference as if fully set forth herein and form the basis for the following cause of action against Juniper.

90. The Accused Products are covered by claim 1 of the '378 Patent.

91. Juniper has directly infringed and continues to infringe claim 1 of the '378 Patent in violation of 35 U.S.C. § 271(a) by, directly or through intermediaries and without Implicit's authority, making, using, selling, or offering to sell the Accused Products in the United States, or importing the Accused Products into the United States.

92. Further and in the alternative, at least since the filing and service of this First Amended Complaint, Juniper has been and now is actively inducing infringement of claim 1 of the '378 Patent in violation of 35 U.S.C. § 271(b). Users of the Accused Products directly infringe claim 1 of the '378 Patent when they use the Accused Products in the ordinary, customary, and intended way. Juniper's inducements include, without limitation and with specific intent to encourage the infringement, knowingly inducing consumers to use the Accused Products within the United States in the ordinary, customary, and intended way by, directly or through intermediaries, supplying the Accused Products to consumers within the United States and instructing such consumers (for example in instructional manuals or videos that Juniper provides online or with the Accused Products) how to use the Accused Products in the ordinary, customary, and intended way, which Juniper knows or should know infringes claim 1 of the '378 Patent. Juniper's inducements may further include, without limitation and with specific intent to encourage the infringement, knowingly inducing manufacturers to make the Accused Products within the United States, or knowingly inducing distributors or resellers to sell or offer to sell the Accused Products within the United States, by, directly or through intermediaries, instructing such

manufacturers, distributors, or resellers to make, sell, or offer to sell the Accused Products in the United States, which Juniper knows or should know infringes claim 1 of the '378 Patent.

93. Further and in the alternative, at least since the filing and service of this First Amended Complaint, Juniper has been and now is actively contributing to infringement of claim 1 of the '378 Patent in violation of 35 U.S.C. § 271(c). Juniper installs, configures, and sells the Accused Products with the Demux Accused Components, each of which is especially made or especially adapted to practice the invention claimed in claim 1 of the '378 Patent. Each Demux Accused Component within the Accused Products constitutes a material part of the claimed invention recited in claim 1 of the '378 Patent and not a staple article or commodity of commerce because it is specifically configured according to claim 1 of the '378 Patent. Juniper's contributions include, without limitation, making, offering to sell, and/or selling within the United States, and/or importing into the United States, the Accused Products, which include one or more Demux Accused Components, knowing each Demux Accused Component to be especially made or especially adapted for use in an infringement of claim 1 of the '378 Patent, and not a staple article or commodity of commerce suitable for substantial noninfringing use.

94. At least as of the filing and service of this First Amended Complaint, Juniper's infringement of the '378 Patent has been and continues to be willful and deliberate.

VII. DAMAGES

95. The allegations of each foregoing paragraph are incorporated by reference as if fully set forth herein.

96. For the above-described infringement, Implicit has been injured and seeks damages to adequately compensate it for Juniper's infringement of the Asserted Patents. Such damages, to be proved at trial, should be no less than the amount of a reasonable royalty under 35 U.S.C. § 284,

together with Implicit's costs and expenses, pre-judgment and post-judgment interest, and supplemental damages for any continuing post-verdict or post-judgment infringement, with an accounting as needed.

97. As set forth above, Juniper's infringement of the Asserted Patents has been and continues to be willful, such that Implicit seeks treble damages under 35 U.S.C. § 284 as appropriate.

98. Juniper's willful infringement of the Asserted Patents renders this case exceptional under 35 U.S.C. § 285, such that Implicit seeks all reasonable attorneys' fees and costs incurred in this litigation, together with pre-judgment and post-judgment interest thereon.

VIII. PRAYER FOR RELIEF

Implicit respectfully requests the following relief:

a. A judgment in favor of Implicit that Juniper has infringed each Asserted Patent, whether literally or under the doctrine of equivalents, as described herein;

b. A judgment and order requiring Juniper to pay Implicit its damages, costs, expenses, and pre-judgment and post-judgment interest for Juniper's infringement of each Asserted Patent as provided under 35 U.S.C. § 284, including supplemental damages for any continuing post-verdict or post-judgment infringement with an accounting as needed;

c. A judgment and order requiring Juniper to pay Implicit enhanced damages for willful infringement as provided under 35 U.S.C. § 284;

d. A judgment and order finding this case exceptional and requiring Juniper to pay Implicit its reasonable attorneys' fees and costs incurred in this litigation pursuant to 35 U.S.C. § 285, together with pre-judgment and post-judgment interest thereon; and

e. Such other and further relief as the Court deems just and proper.

IX. JURY DEMAND

Pursuant to Federal Rule of Civil Procedure 38(b), Implicit requests a jury trial of all issues triable of right by a jury.

Dated: March 19, 2019

Respectfully submitted,

By: /s/ William E. Davis, III
William E. Davis, III
Texas State Bar No. 24047416
bdavis@bdavisfirm.com
Christian J. Hurt
Texas State Bar No. 24059987
churt@bdavisfirm.com
Edward Chin (Of Counsel)
Texas State Bar No. 50511688
echin@bdavisfirm.com
Debra Coleman (Of Counsel)
Texas State Bar No. 24059595
dcoleman@bdavisfirm.com
The Davis Firm, PC
213 N. Fredonia Street, Suite 230
Longview, Texas 75601
Telephone: (903) 230-9090
Facsimile: (903) 230-9661

Spencer Hosie (admitted *pro hac vice*)
California State Bar No. 101777
shosie@hosielaw.com
Brandon C. Martin (admitted *pro hac vice*)
California State Bar No. 269624
bmartin@hosielaw.com
Hosie Rice LLP
600 Montgomery Street, 34th Floor
San Francisco, California 94111
Telephone: (415) 247-6000
Facsimile: (415) 247-6001

Counsel for Plaintiff Implicit, LLC

CERTIFICATE OF SERVICE

The undersigned certifies that the foregoing document and all attachments thereto are being filed electronically in compliance with Local Rule CV-5(a) on this March 19, 2019. As such, this document is being served on all counsel, each of whom is deemed to have consented to electronic service. Local Rule CV-5(a)(3)(V).

/s/ William E. Davis, III
William E. Davis, III

EXHIBIT 1

US008056075B2

(12) United States Patent
Balassanian

(10) **Patent No.: US 8,056,075 B2**
(45) **Date of Patent: *Nov. 8, 2011**

(54) **SERVER REQUEST MANAGEMENT**

(76) Inventor: **Edward Balassanian**, Kirkland, WA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 931 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **11/933,161**

(22) Filed: **Oct. 31, 2007**

(65) **Prior Publication Data**

US 2008/0140772 A1 Jun. 12, 2008

Related U.S. Application Data

(63) Continuation of application No. 11/241,985, filed on Oct. 4, 2005, now Pat. No. 7,774,740, which is a continuation of application No. 09/968,704, filed on Oct. 1, 2001, now Pat. No. 6,976,248, which is a continuation of application No. 09/040,972, filed on Mar. 18, 1998, now Pat. No. 6,324,685.

(51) **Int. Cl.**
G06F 9/445 (2006.01)
G06F 9/44 (2006.01)

(52) **U.S. Cl.** **717/177**; 717/171; 709/203

(58) **Field of Classification Search** 717/168–177; 709/203–204

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,920,725	A *	7/1999	Ma et al.	717/171
5,923,885	A *	7/1999	Johnson et al.	717/176
5,926,631	A *	7/1999	McGarvey	703/23
6,105,063	A *	8/2000	Hayes, Jr.	709/223
6,195,794	B1 *	2/2001	Buxton	717/108
6,253,228	B1 *	6/2001	Ferris et al.	709/203
6,757,894	B2 *	6/2004	Eylon et al.	717/177
6,766,366	B1 *	7/2004	Schafer et al.	709/223
6,996,817	B2 *	2/2006	Birum et al.	717/170
7,069,294	B2 *	6/2006	Clough et al.	709/203
7,131,122	B1 *	10/2006	Lakhdhir	717/168
7,150,015	B2 *	12/2006	Pace et al.	717/176
7,155,715	B1 *	12/2006	Cui et al.	717/177
7,444,629	B2 *	10/2008	Chirakansakcharoen et al.	717/166
7,523,158	B1 *	4/2009	Nickerson et al.	709/203
7,721,283	B2 *	5/2010	Kovachka-Dimitrova et al.	717/177
7,814,475	B2 *	10/2010	Cohen et al.	717/168
7,934,212	B2 *	4/2011	Lakhdhir	717/170
7,991,834	B2 *	8/2011	Ferris et al.	709/203

OTHER PUBLICATIONS

Bonisch et al, "Server side compresslets for internet multimedia streams", IEEE, pp. 82-86, 1999.*

Wirthlin et al, "Web based IP evaluation and distribution using applets", IEEE vol. 22, No. 8, pp. 985-994, 2003.*

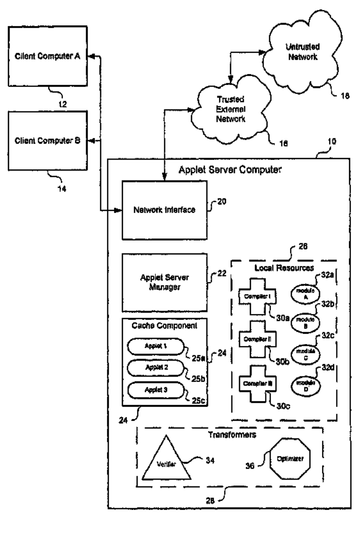
(Continued)

Primary Examiner — Anil Khatri

(57) **ABSTRACT**

The present invention is an applet server which accepts requests for applets from client computers. A request specifies the format in which an applet is to be delivered to the requesting client computer. The applet server has a cache which it uses to store applets for distribution to client computers. If the specified form of the requested applet is available in the cache, the applet server transmits the applet to the requesting client. If the applet is not available in the cache, the server will attempt to build the applet from local resources (program code modules and compilers) and transformer programs (verifiers and optimizers). If the applet server is able to build the requested applet, it will then transmit the applet to the requesting client computer. If the applet server is unable to build the requested applet, it will pass the request to another applet server on the network for fulfillment of the request.

13 Claims, 3 Drawing Sheets



OTHER PUBLICATIONS

Ding et al, "Selective Java applet filtering on Internet", IEEE, pp. 110-114, 1999.*

Lai et al, "On the performance of wide area thin client computing", ACM Trans. on Compt. Sys> vo. 24, No. 2, pp. 175-209, 2006.*

* cited by examiner

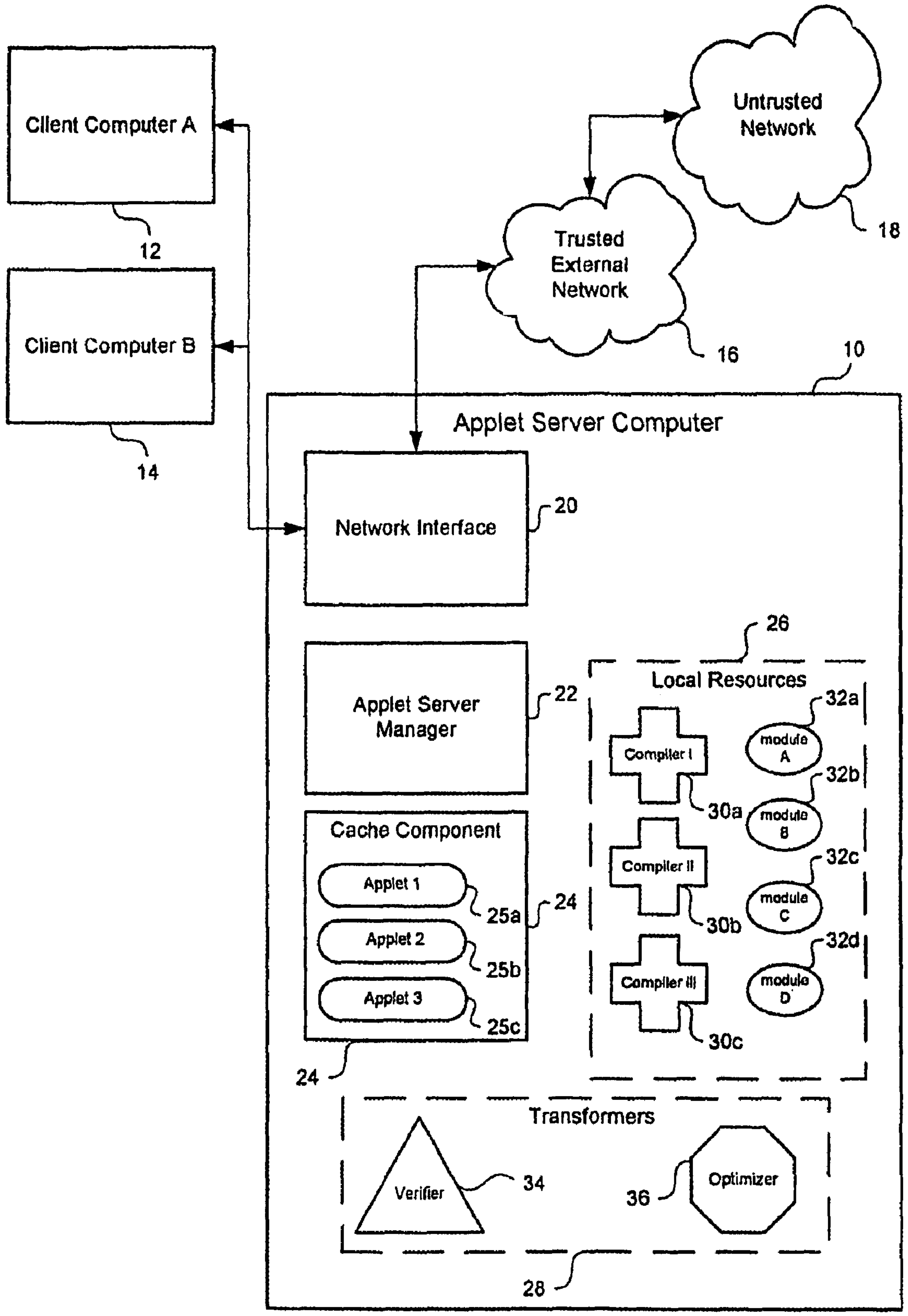

Client Computer A
12
Client Computer B
14
Untrusted Network
18
Trusted External Network
16
10
Applet Server Computer
Network Interface
20
26
Local Resources
Applet Server Manager
22
32a
Compiler I
module A
32b
30a
module B
32c
Compiler II
module C
Cache Component
Applet 1
25a
24
Applet 2
25b
30b
32d
Compiler III
module D
Applet 3
25c
30c
24
Transformers
Verifier
34
36
Optimizer
28

Fig. 1

Request Data Type

Tag	Value
Applet-URL	(String) specifies the name of the requested applet
Code-Type	(Source/Intermediate/Binary) specifies the format the applet is to be delivered to the requesting client in. A request for binary would specify the CPU of the requesting client (e.g., x86)
Verification-Level	(0-100) specifies the degree of verification to be performed. 0 = no/minimal verification, 100 = maximum verification (highest level of security).
Optimization-Level	(0-100) specifies the degree of optimization to be performed. 0 = no/minimal optimization, 100 = maximum optimization.

Fig. 2A

Code Data Type

Tag	Value
Applet-URL	(String) specifies the name of the requested applet
Code-Type	(Source/Intermediate/Binary) specifies the format the applet is to be delivered to the requesting client in. A request for binary would specity the CPU of the requesting client (e.g., x86)
Verification-Level	(0-100) specifies the degree of verification to be performed. 0 = no/minimal verification, 100 = maximum verification (highest level of security).
Optimization-Level	(0-100) specifies the degree of optimization to be performed. 0 = no/minimal optimization, 100 = maximum optimization.
Applet Length	(0-2^{32}) specifies the size of the requested applet.
Applet Code	The Requested Applet in the form specified by the request data type.

Fig. 2B

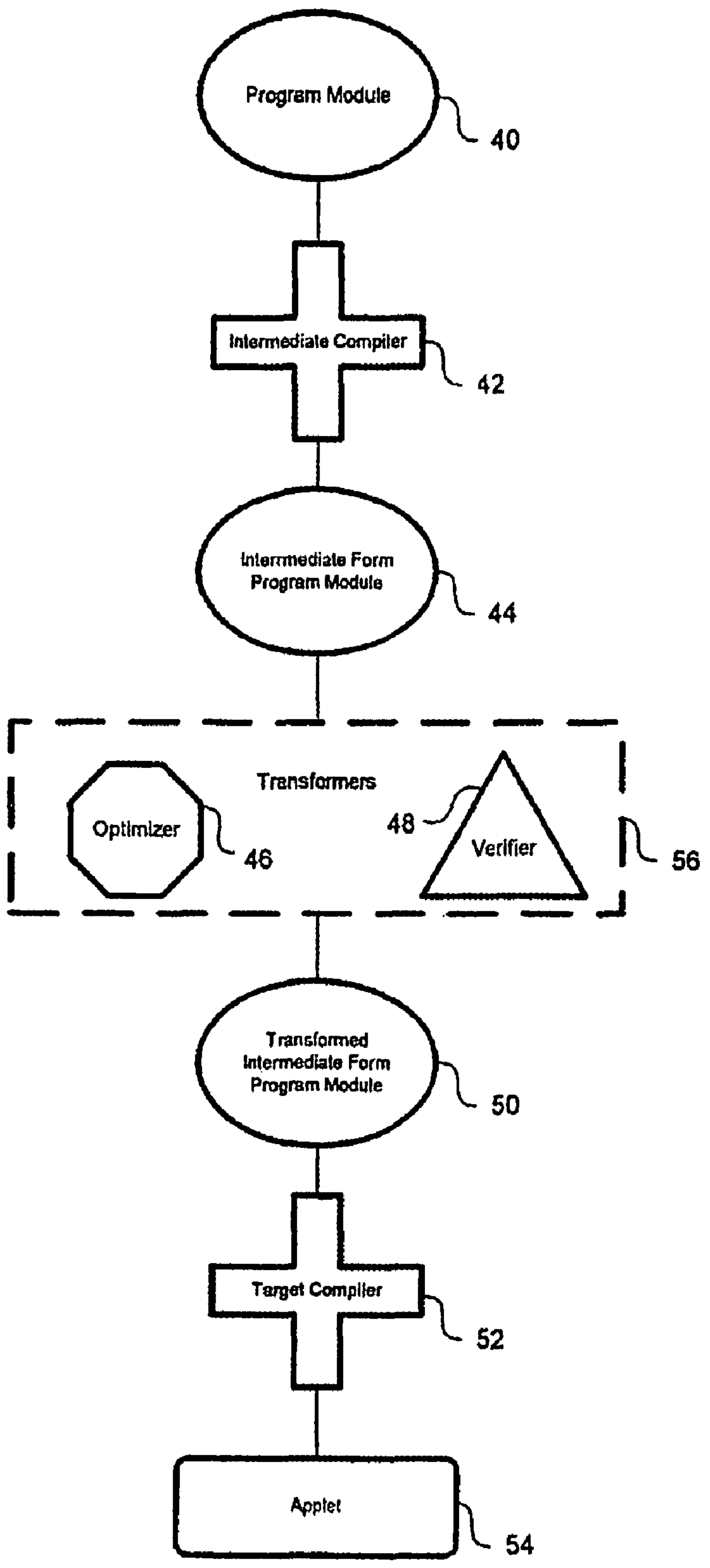

Program Module
40
Intermediate Compiler
42
Intermediate Form Program Module
44
Transformers
Optimizer
46
48
Verifier
56
Transformed Intermediate Form Program Module
50
Target Compiler
52
Applet
54

Fig. 3

SERVER REQUEST MANAGEMENT

PRIORITY CLAIM

This application is a continuation of U.S. application Ser. No. 11/241,985 filed Oct. 4, 2005 now U.S. Pat. No. 7,774,740 which is a continuation of U.S. application Ser. No. 09/968,704 filed on Oct. 1, 2001 now U.S. Pat. No. 6,976,248, which is a continuation of U.S. application Ser. No. 09/040,972 filed on Mar. 18, 1998, now U.S. Pat. No. 6,324,685.

BACKGROUND OF THE INVENTION

Field of the Invention

The present invention relates to computer operating system and, in particular to a server architecture providing application caching and security verification.

The growth of the Internet's importance to business, along with the increased dependence upon corporate networks, has created a demand for more secure and efficient computer systems. The traditional solution to this problem has been to depend upon improvements in hardware performance to make up for the performance penalty that is typically incurred when a computer system is made more secure and stable. Increased interconnectivity has also created a need for improved interoperability amongst a variety of computers that are now connected to one another. One solution to the problem of the variety of computers interconnected via the Internet and corporate networks has been the development of portable architecture neutral programming languages. The most widely known of these is the Java™ programming language, though, there are numerous other architecture neutral languages.

Architecture neutral programming languages allow programs downloaded from a server computer to a client computer to be interpreted and executed locally. This is possible because the compiler generates partially compiled intermediate byte-code, rather than fully compiled native machine code. In order to run a program, the client machine uses an interpreter to execute the compiled byte-code. The byte-codes provide an architecture neutral object file format, which allows the code to be transported to multiple platforms. This allows the program to be run on any system which implements the appropriate interpreter and run-time system. Collectively, the interpreter and runtime system implement a virtual machine. This structure results in a very secure language.

The security of this system is premised on the ability of the byte-code to be verified independently by the client computer. Using the Java™ programming language or some other virtual machine implementing technology, a client can ensure that the downloaded program will not crash the user's computer or perform operations for which it does not have permission.

The traditional implementations of architecture neutral languages are not without problems. While providing tremendous cross platform support, the current implementations of architecture neutral languages require that every client performs its own verification and interpretation of the intermediate code. The high computation and memory requirements of a verifier, compiler and interpreter restrict the applicability of these technologies to powerful client computers.

Another problem with performing the verification process on the client computer is that any individual within an organization may disable some or all of the checks performed on downloaded code. The current structure of these systems makes security management at the enterprise level almost impossible. Since upgrades of security checking software must be made on every client computer, the cost and time involved in doing such upgrades makes it likely that outdated or corrupt copies of the verifier or interpreter exist within an organization. Even when an organization is diligent in maintaining a client based security model, the size of the undertaking in a large organization increases the likelihood that there will be problems.

There is a need for a scalable distributed system architecture that provides a mechanism for client computers to request and execute applets in a safe manner without requiring the client machines to have local resources to compile or verify the code. There is a further need for a system in which the applets may be cached in either an intermediate architecture neutral form or machine specific form in order to increase overall system performance and efficiency.

SUMMARY OF THE INVENTION

In accordance with one embodiment of the invention, an applet server architecture is taught which allows client computers to request and execute applets in a safe manner without requiring the client to have local resources to verify or compile the applet code. Compilation and byte-code verification in the present invention are server based and thereby provide more efficient use of resources and a flexible mechanism for instituting enterprise-wide security policies. The server architecture also provides a cache for applets, allowing clients to receive applet code without having to access nodes outside the local network. The cache also provides a mechanism for avoiding repeated verification and compilation of previously requested applet code since any client requesting a given applet will have the request satisfied by a single cache entry.

Machine specific binary code is essentially interpreted code since the processor for a given computer can essentially be viewed as a form of an interpreter, interpreting binary code into the associated electronic equivalents. The present invention adds a level of indirection in the form of an intermediate language that is processor independent. The intermediate language serves as the basis for security verification, code optimizations, or any other compile time modifications that might be necessary. The intermediate form allows a single version of the source to be stored for many target platforms instead of having a different binary for each potential target computer. Compilations to the target form can either be done at the time of a cache hit or they can be avoided all together if the target machine is able to directly interpret the intermediate form. If the compilation is done on the server, then a copy of the of the compiled code as well as the intermediate form can be stored in the cache. The performance advantage derived from caching the compiled form as well as the intermediate depends upon the number of clients with the same CPU.

The novel features believed characteristic of the invention are set forth in the appended claims. The invention itself, however, as well as other features and advantages thereof will best be understood by reference to the detailed description which follows, when read in conjunction with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is diagram showing the major components which may be used to implement an applet server in one embodiment of the present invention;

FIG. **2***a* is a table which illustrates the structure of the request format data type;

FIG. **2***b* is a table which illustrates the structure of the returned code data type.

FIG. **3** is a diagram showing the compilation and transformation of a program module into an applet in a particular form.

DETAILED DESCRIPTION OF THE INVENTION

Referring to FIG. **1**, an applet server architecture according to one embodiment of the invention is based on an applet server computer **10** which in turn is connected to client computer A **12**, client computer B **14**, an external network **16** and an untrusted network **18**. The applet server computer **10** connects to client computers **12** and **14**, an external network **16**, and an untrusted network **18** by means of a network interface **20**. Typically this connection will involve one or more of the computers or networks having a connection to the Internet.

The applet server computer **10** accomplishes its objectives by manipulating computer programs in several formats. An applet (e.g. applets **1**-**3**, **25***a*-**25***c*) is any form of program instructions, whether in binary, source or intermediate format. In the case of this architecture, the applet code can either be a self contained program, or it can be a code fragment associated with a larger application.

Binary format refers to processor specific machine instructions suitable for running natively on a given computing platform (also referred to as "target" because of the concept of "targeting" a compiler to produce binary code for a given processor type).

Source refers to non-binary applet code, generally in the form of high level languages (i.e. the C™, C++™, Java™, Visual Basic™, ActiveX™, Fortran™, and Modula™ programming languages.

Intermediate format refers to a common intermediate byte-code that is produced by compiling a given source code input. The intermediate byte-code need not necessarily be Java™ byte-code.

Treating applets in this general sense allows client computers **12** and **14** to request not only applications, but portions of applications. Client computers **12** and **14** are thus able to use applet server computer **10** as the equivalent of a loader, loading in appropriate parts of the application in the form of applets. In turn client computers **12** and **14** can run large applications without requiring that the client computers **12** and **14** have the resources to store the entire application in memory at once.

Having the applets delivered from applet server computer **10** allows code in intermediate form to be verified, optimized, and compiled before being transmitted to client computers **12** and **14**. This reduces the amount of work the client computers **12** and **14** have to do and provides a convenient way to impose global restrictions on code.

In operation, client computer A **12** transmits a request to an applet server computer **10** requesting an applet in a particular form. The form may be selected from a large matrix of many possible forms that can be recognized by the system. The request specifies the format (source, intermediate, or binary) in which the client wishes to receive the applet. The request may also specify that the applet be verified or have some other transformation operation preformed upon it. Verification, optimization and compression are examples of types of transformation operations. The request is received by the network interface **20** of the applet server computer **10** which passes the request onto the applet server manager **22**.

After interpreting the request, the applet server manager **22** checks to see if the requested applet is available in the cache **24**. The cache **24** stores applets in a variety of formats (source, intermediate, or binary). If the requested form of the applet is available in the cache **24** (applet **1** **25***a*, applet **2** **25***b*, or applet **3** **25***c* in this example) the applet server manager **22** instructs the network interface **20** to transmit the applet to requesting client computer A **12**.

If the requested applet is not available in the cache **24**, then the applet server manager **22** will attempt to build the requested applet from local resources **26** and one or more transformation operations performed by one or more of the transformers **28**. Local resources **26** are comprised of compilers **30***a*, **30***b* and **30***c* and program code modules **32***a*, **32***b*, **32***c* and **32***d*. The requested applet is built by selecting one or more program code modules **32** and compiling them with one or more compilers **30**. Transformer operations may be performed by the verifier **34** or the optimizer **36**. After the applet server manager **22** builds the applet, the network interface **20** transmits the applet to the requesting client computer A **12**.

If the request can not be satisfied by building the applet from local resources **26** and transformers **28**, the applet server manager **22** will pass a request for the requested applet to external network **16** and/or untrusted network **18**. The applet server manager **22** may request the applet in intermediate form or in executable form or it may request the local resources **26** and transformers **28** it needs to complete building the applet itself.

The cache **24** is capable of responding to the following commands: GET, PUT, and FLUSH. GET is used to retrieve a given applet from the cache. PUT is used to store an applet in the cache. FLUSH is used to clear the cache of one or more entries. When the cache is unable to locate an item in response to a GET operation, it returns a cache miss. The program which issued the GET command is then responsible for locating the desired form of the applet by other means and optionally storing it in the cache when it is retrieved (using the PUT operation). The FLUSH command will clear the cache of one or more entries and any subsequent GETs for the FLUSHed applet code will result in a cache miss. This is useful if a particular applet needs to be updated from a remote server on a periodic basis. When using PUT, the program issuing the command specifies a time to live (TTL) in the cache. When the TTL expires, the cache entry is removed by means of a FLUSH operation.

Local resources **26** are comprised of program modules **32** (applets in source form, not the requested form) and compilers **30**. The program modules **32** are run through the compilers **30** in order to produce applets in the requested form. The applet server manager **20** may also direct the modules **32** to be processed by a verifier **34** or another transformer such as an optimizer **36**. Program modules **32** are program code used to build applets. Program modules **32** may be stored in local resources **26** in source binary, or intermediate formats. When an applet is built it may require the operation of one or more compilers **30** upon one or more program modules **32**. The program modules **32** may be combined and recompiled with previously cached applets and the resulting applet may be also cached for use at a future time. Additionally, program modules **32**, compilers **30** and transformers **28** (including verifiers **34** and optimizers **36**) may be distributed across a network. The applet server manager **22** may pass requests for the components it needs to build a particular applet back to the network interface **20** which in turn passes the request onto the rest of the network and may include external network **16** and untrusted network **18**.

FIG. **3** provides further illustration of how an applet is produced from local resources and transformers. In this illustration the request is for an optimized and verified applet compiled to a machine specific form. A program module **40** is compiled into an intermediate form program module **44** by an intermediate compiler **42**. The intermediate form program module **44** is then transformed by an optimizer **46** or a verifier **48**. The resulting transformed intermediate form program module **50** is then compiled by target compiler **52** into machine specific code applet **54**.

There are two types of compilers used to build applets: intermediate compilers **42** and target compilers **52**. The intermediate compiler **42** compiles program modules (source applet code) **40** and produces a common intermediate pseudo-binary representation of the source applet code (intermediate form program module **44**). The word pseudo is used because the intermediate form **44** is not processor specific but is still a binary representation of the source program module **40**. This intermediate form can be re-targeted and compiled for a particular processor. Alternatively, the intermediate form **44** can be interpreted by an interpreter or virtual machine that understands the internal binary representation of the intermediate form. A target compiler **52** compiles intermediate applet code **44** into an applet **54** in a processor specific format (binary) suitable for running natively on a given computing platform.

Transformers **56** are programs that take in intermediate code and put out intermediate code. Transformers **56** are generally used for things like verification and optimization. Other transformers might included compressors that identify portions of code that can be replaced with smaller equivalents. Transformers can be matched up to any other component that takes in intermediate code as an input. These include the cache **24** and the target compilers **52**. Global policies for transformers **56** can be implemented which ensure that all applets are run through some set of transformers before being returned to the client.

A verifier **48** is a type of transformer that is able to analyze input code and determine areas that might not be safe. The verifier **48** can determine the level of safety. Some verifiers **48** look for areas where unsafe or protected memory is being accessed, others might look for accesses to system resources such as IO devices. Once a verifier **48** determines the portion of unsafe applet code several steps can be taken. The offending code portion can be encased with new code that specifically prevents this unsafe code section from being executed. The unsafe code can be modified to be safe. The unsafe code can be flagged in such a way that a user can be warned about the possible risks of executing the code fragment. The verifier's role can therefore be summarized as determining where unsafe code exists and possibly altering the offending code to render it harmless. Verifiers **48** can operate on any format of input code, whether in source intermediate or binary form. However, since intermediate code is a common format, it is most efficient to have a single verifier that will operate on code in this format. This eliminates the need to build specific knowledge of various source languages into the verifier. Verifiers **48** are a form of a transformer. Verifiers **48** take in intermediate code and put out verified intermediate code. Verifiers **48** are responsible for identifying non-secure portions of code in the intermediate code and modifying this code to make it secure. Security problems generally include access to memory areas that are unsafe (such as system memory, or memory outside the application space of the applet).

The choice of adding in the verification step can be left up to the client computers **12**, the applet server computer **10** (see FIG. **1**), or can be based on the network that the applet originated from. Server managers can institute global policies that affect all clients by forcing all applets to be run through the verifier **48**. Alternatively, verification can be reserved for un-trusted networks (**18** in FIG. **1**), or it can be left up to the

client to determine whether the verification should be performed. In the preferred embodiment, verification level is determined by the applet server **10**. In this way, a uniform security policy may be implemented from a single machine (i.e., the applet server **10**).

Optimizers **46** are another type of transformer program. Optimizers **46** analyze code, making improvements to well known code fragments by substituting in optimized but equivalent code fragments. Optimizers **46** take in intermediate code **44** and put out transformed intermediate code **50**. The transformed intermediate code **50** is functionally equivalent to the source intermediate code **44** in that they share the same structure.

Referring again to FIG. **1**, policies may be instituted on the applet server **10** that force a certain set of request parameters regardless of what the client asked for. For example, the applet server manager **22** can run the applet through a verifier **34** or optimizer **36** regardless of whether the client **12** requested this or not. Since the server **10** might have to go to an untrusted network **18** to retrieve a given applet, it will then run this applet through the required transformers **28**, particularly the verifier **34** before returning it to the client **12**. Since clients **12** and **14** have to go through the applet server computer **10**, this ensures that clients **12** and **14** do not receive applets directly from an untrusted network **18**. In addition, since the server will be dealing directly with untrusted network **18**, it can be set up to institute policies based on the network. A trusted external network **16** may be treated differently than an untrusted network **18**. This will provide the ability to run a verifier **34** only when dealing with an untrusted network **18**, but not when dealing with a trusted external network **16**. In one embodiment, all intermediate code is passed through a verifier **34** and the source of the code merely determines the level of verification applied.

The client **12** is the target computer on which the user wishes to execute an applet. The client **12** requests applets from the server **10** in a specific form. Applets can be requested in various formats including source, intermediate and binary. In addition, an applet can be requested with verification and/or other compile time operations. Optionally, the client **12** can pass a verifier to the server to provide verification. If the server **10** implements its own security, then both the client and server verifiers will be run. The verifier that is passed from the client to the server is cached at the server for subsequent verification. The client can refer to this verifier by a server-generated handle to avoid having to pass the verifier each time an applet is requested.

Client computers **12** and **14** requesting applet code in intermediate format need to have an interpreter or virtual machine capable of interpreting the binary code in the intermediate format if the applet is to be executed on the client machine.

In the preferred embodiment, requests to the applet server are in a format similar to those of an HTTP header and are comprised of tags and values. In one embodiment, an HTTP GET method is used to make the request (though use of the HTTP protocol is not necessary to implement the present invention). The request is made up of a series of tags which specify the requested applet, the platform on which it is to be run and the type of code (source/intermediate/binary), a verification level and an optimization level. New tags and values can be added to extend functionality as needed and the applet server manager **22** will discard any tag it does not recognize. When the applet server computer **10** returns the requested applet to the requesting client computer A **12**, it will transmit the request header followed by the applet code. In this instance, the header will additionally include a field which defines the length of the applet code. FIG. **2** provides a table which illustrates the request format and the returned code format.

While this invention has been described with reference to specific embodiments, this description is not meant to limit the scope of the invention. Various modifications of the disclosed embodiments, as well as other embodiments of the invention, will be apparent to persons skilled in the art upon reference to this description. It is therefore contemplated that the appended claims will cover any such modifications or embodiments as fall within the scope of the invention.

I claim:

1. A method for delivering one or more applets to one or more client computers, comprising, in no particular order, the steps of:

configuring an applet server manager at a server computer to manage at least one request from the one or more client computers for the one or more applets, the applet server manager having access to one or more networks;

receiving the at least one request at the applet server manager;

passing the at least one request from the applet server manager to at least one of the one or more networks;

receiving the one or more applets at the applet server manager from the at least one of the one or more networks;

processing the one or more applets at the applet server manager, wherein processing the one or more applets includes at least one of the following steps:

compressing the one or more applets before sending the one or more applets to the one or more client computers,

optimizing the one or more applets before sending the one or more applets to the one or more client computers, and

verifying the one or more applets before sending the one or more applets to the one or more client computers; and

sending the one or more applets from the applet server manager to the one or more client computers.

2. The method of claim **1** wherein processing the one or more applets further includes discarding at least one of the one or more applets based on one or more policies.

3. The method of claim **1** wherein processing the one or more applets further includes transforming at least one of the one or more applets by modifying the data in the at least one of the one or more applets before sending the one or more applets to the one or more client computers.

4. The method of claim **1** wherein at least one of the one or more networks is a trusted network.

5. The method of claim **1** wherein at least one of the one or more networks is an untrusted network.

6. The method of claim **1** wherein the at least one request specifies the name of at least one of the one or more applets using a Uniform Resource Locator.

7. The method of claim **1** wherein the at least one request specifies at least one of the one or more applets using an HTTP header.

8. The method of claim **1** wherein the at least one request specifies at least one of the one or more applets using a cookie.

9. The method of claim **1** wherein processing the one or more applets includes producing at least one of the one or more applets in the form of a web page.

10. The method of claim **1** wherein processing the one or more applets includes producing at least one of the one or more applets in the form of a portion of a web page.

11. The method of claim **1** wherein sending the one or more applets includes sending at least one of the one or more applets to the one or more client computers in an HTTP response.

12. The method of claim **1** wherein the at least one request is processed prior to passing the at least one request from the applet server manager to at least one of the one or more networks.

13. The method of claim **1** wherein processing the one or more applets further includes performing security verification based on one or more policies.

* * * * *

EXHIBIT 2

US008856779B2

(12) **United States Patent**
Balassanian

(10) **Patent No.: US 8,856,779 B2**
(45) **Date of Patent: *Oct. 7, 2014**

(54) APPLICATION SERVER FOR DELIVERING APPLETS TO CLIENT COMPUTING DEVICES IN A DISTRIBUTED ENVIRONMENT

(75) Inventor: **Edward Balassanian**, Kirkland, WA (US)

(73) Assignee: **Implicit, LLC**, Seattle, WA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 1 day.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **13/269,905**

(22) Filed: **Oct. 10, 2011**

(65) **Prior Publication Data**

US 2013/0042228 A1 Feb. 14, 2013

Related U.S. Application Data

(63) Continuation of application No. 11/933,161, filed on Oct. 31, 2007, now Pat. No. 8,056,075, which is a continuation of application No. 11/241,985, filed on Oct. 4, 2005, now Pat. No. 7,774,740, which is a continuation of application No. 09/968,704, filed on Oct. 1, 2001, now Pat. No. 6,976,248, which is a continuation of application No. 09/040,972, filed on Mar. 18, 1998, now Pat. No. 6,324,685.

(51) **Int. Cl.**
G06F 9/445 (2006.01)

(52) **U.S. Cl.**
USPC **717/177**; 717/175; 709/203

(58) **Field of Classification Search**
USPC 717/100–103, 120–121, 169–178; 709/203–204
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,706,502	A	1/1998	Foley et al.
5,761,421	A	6/1998	van Hoff et al.
5,805,829	A	9/1998	Cohen et al.
5,828,840	A	10/1998	Cowan et al.
5,835,712	A	11/1998	DuFresne
5,848,274	A	12/1998	Hamby et al.
5,872,915	A	2/1999	Dykes et al.
5,884,078	A	3/1999	Faustini

(Continued)

OTHER PUBLICATIONS

Schlumberger et al, "Jarhead Analysis and Detection of Malicious Java Applets", ACM, 249-258, 2012.*

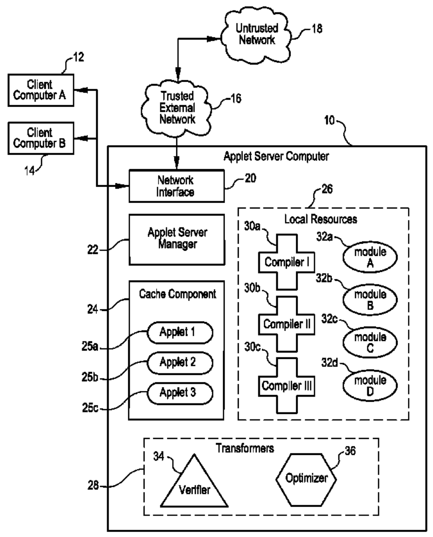
(Continued)

Primary Examiner — Anil Khatri
(74) *Attorney, Agent, or Firm* — Meyertons, Hood, Kivlin, Kowert & Goetzel, P.C.

(57) **ABSTRACT**

An applet server accepts requests for applets from client computers. A request specifies the format in which an applet is to be delivered to the requesting client computer. The applet server has a cache used to store applets for distribution to client computers. If the specified form of the requested applet is available in the cache, the applet server transmits the applet to the requesting client. If the applet is not available in the cache, the server will attempt to build the applet from local resources (program code modules and compilers) and transformer programs (verifiers and optimizers). If the applet server is able to build the requested applet, it will transmit the applet to the requesting client computer. If the applet server is unable to build the requested applet, it will pass the request to another applet server on the network for fulfillment of the request.

20 Claims, 3 Drawing Sheets



(56) **References Cited**

U.S. PATENT DOCUMENTS

5,911,776	A	6/1999	Guck	
5,920,725	A	7/1999	Ma et al.	
5,923,885	A	7/1999	Johnson et al.	
5,926,631	A	7/1999	McGarvey	
5,943,496	A	8/1999	Li et al.	
5,944,784	A *	8/1999	Simonoff et al.	709/203
5,996,022	A	11/1999	Krueger et al.	
6,105,063	A	8/2000	Hayes, Jr.	
6,195,794	B1	2/2001	Buxton	
6,212,673	B1 *	4/2001	House et al.	717/100
6,230,184	B1	5/2001	White et al.	
6,253,228	B1	6/2001	Ferris et al.	
6,279,151	B1	8/2001	Breslau et al.	
6,282,702	B1	8/2001	Ungar	
6,295,643	B1	9/2001	Brown et al.	
6,317,781	B1	11/2001	De Boor et al.	
6,321,377	B1	11/2001	Beadle et al.	
6,324,685	B1	11/2001	Balassanian	
6,327,701	B2	12/2001	Ungar	
6,330,710	B1	12/2001	O'Neil et al.	
6,434,745	B1 *	8/2002	Conley et al.	717/177
6,446,081	B1	9/2002	Preston	
6,502,236	B1	12/2002	Allen et al.	
6,546,554	B1 *	4/2003	Schmidt et al.	717/176
6,594,820	B1	7/2003	Ungar	
6,611,858	B1 *	8/2003	Aravamudan et al.	709/203
6,636,900	B2	10/2003	Abdelnur	
6,643,683	B1	11/2003	Drumm et al.	
6,704,926	B1	3/2004	Blandy et al.	
6,718,364	B2 *	4/2004	Connelly et al.	709/203
6,718,540	B1	4/2004	Azua et al.	
6,741,608	B1	5/2004	Bouis et al.	
6,742,165	B2	5/2004	Lev et al.	
6,745,386	B1	6/2004	Yellin	
6,754,693	B1 *	6/2004	Roberts et al.	709/205
6,757,894	B2	6/2004	Eylon et al.	
6,766,366	B1	7/2004	Schafer et al.	
6,772,408	B1	8/2004	Velonis et al.	
6,789,252	B1 *	9/2004	Burke et al.	717/100
6,802,061	B1 *	10/2004	Partovi et al.	717/173
6,832,263	B2	12/2004	Polizzi et al.	
6,836,889	B1	12/2004	Chan et al.	
6,842,897	B1	1/2005	Beadle et al.	
6,865,732	B1	3/2005	Morgan	
6,865,735	B1	3/2005	Sirer et al.	
6,910,128	B1 *	6/2005	Skibbie et al.	713/170
6,947,943	B2 *	9/2005	DeAnna et al.	717/120
6,950,850	B1 *	9/2005	Leff et al.	709/203
6,976,248	B2	12/2005	Balassanian	
6,993,743	B2 *	1/2006	Crupi et al.	717/102
6,996,817	B2	2/2006	Birum et al.	
7,051,315	B2 *	5/2006	Artzi et al.	717/103
7,069,294	B2	6/2006	Clough et al.	
7,127,700	B2	10/2006	Large	
7,131,111	B2 *	10/2006	Passanisi	717/121
7,131,122	B1	10/2006	Lakhdhir	
7,136,896	B1	11/2006	Srinivas et al.	
7,150,015	B2 *	12/2006	Pace et al.	717/176
7,155,715	B1	12/2006	Cui et al.	
7,346,655	B2	3/2008	Donoho et al.	
7,415,706	B1 *	8/2008	Raju et al.	717/170
7,434,215	B2 *	10/2008	Boykin et al.	717/169
7,444,629	B2	10/2008	Chirakansakcharoen et al.	
7,472,171	B2 *	12/2008	Miller et al.	709/219
7,493,591	B2	2/2009	Charisius et al.	
7,519,684	B2	4/2009	Backhouse et al.	
7,523,158	B1	4/2009	Nickerson et al.	
7,530,050	B2	5/2009	Mohan et al.	
7,562,346	B2	7/2009	Jhanwar et al.	
7,590,643	B2 *	9/2009	Demiroski et al.	1/1
7,624,394	B1 *	11/2009	Christopher, Jr.	717/177
7,703,093	B2 *	4/2010	Fischer et al.	717/177
7,707,571	B1 *	4/2010	Harris et al.	717/177
7,721,283	B2 *	5/2010	Kovachka-Dimitrova et al.	717/177
7,730,482	B2 *	6/2010	Illowsky et al.	717/177
7,774,742	B2 *	8/2010	Gupta et al.	717/101
7,814,475	B2	10/2010	Cohen et al.	
7,934,212	B2	4/2011	Lakhdhir	
7,991,834	B2	8/2011	Ferris et al.	
8,056,075	B2 *	11/2011	Balassanian	717/177
8,127,274	B2 *	2/2012	Astheimer	717/120
8,285,777	B2 *	10/2012	Giles et al.	709/203
8,392,906	B2 *	3/2013	Broussard et al.	717/170
8,392,912	B2 *	3/2013	Davis et al.	717/177
8,490,082	B2 *	7/2013	Moore et al.	717/175
8,499,278	B2 *	7/2013	Hughes	717/101
8,615,545	B1 *	12/2013	Lakhdhir	709/203

OTHER PUBLICATIONS

Choi et al, "Efficient Execution of Application Applets Based on Persistent Object Caching in Java Card System" ACM 268-272, 2009.*
Yang et al, "Developing Integrated Web and Database Applications a Using JAVA Applets and JDBC Drivers" ACM, pp. 302-206, 1998.*
Roberts, "Resurrecting the Applet Paradigm", ACM, pp. 521-525, 2007.*
Defendant Microsoft Corporation's Invalidity Contentions Pursuant to Patent L.R. 3-3, Implicit Networks, Inc., (Plaintiff) vs. Sybase, Inc. and Microsoft Corporation, (Defendants), United States District Court Northern District of California San Francisco Division (Oct. 1, 2009), including Exhibits A & B, 12 pages. [Case No. 09-cv-01478].
Eric A. Meyer and Peter Murray, "Borealis Image Server," In Proceedings of the Fifth International World Wide Web Conference; Computer Networks and ISDN Systems, V. 28, Nos. 7-11, May 6-10, 1996, pp. 1123-1137. [Retrieved from http://meyerweb.com/eric/talks/www5/borealis.html 3/18/14].
Marc H. Brown and Marc A. Najork, "Distributed Active Objects," In Proceedings of the Fifth International World Wide Web Conference; Computer Networks and ISDN Systems, V. 28, Nos. 7-11, May 6-10, 1996, pp. 1037-1052. [Retrieved from ftp://gatekeeperresearch.compaq.com/pub/dec/SRC/research-reports/SRC-141a.html 3/18/14].
Michael P. Plezbert and Ron K. Cytron, "Does "Just in Time" = "Better Late than Never"?" In Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Jan. 15-17, 1997, pp. 120-131.
Anawat Chankhunthod, et al., "A Hierarchical Internet Object Cache," In Proceedings of the Annual Technical Conference on USENIX 1996, Jan. 22-26, 1996, 11 pages.
E.P. Markatos, E.P., "Main Memory Caching of Web Documents," In Proceedings of the Fifth International World Wide Web Conference; Computer Networks and ISDN Systems, V. 28, Nos. 7-11, May 6-10, 1996, 15 pages. [Retrieved from http://archvlsi.ics.forth.gr/papers/www5/Overview.html 3/18/14].
M. Frans Kaashoek, et al., "Server Operating Systems," In Proceedings of the 7th Workshop on ACM SIGOPS European Workshop: Systems Support for Worldwide Applications, Sep. 9-11, 1996, 8 pages.
Barron C. House!, et al., "WebExpress: A System for Optimizing Web Browsing in a Wireless Environment," In Proceedings of the 2nd Annual International Conference on Mobile Computing and Networking, Nov. 1996, 9 pages.
Thomas T. Kwan, et al., "NCSA's World Wide Web Server: Design and Performance," Computer, V. 28 No. 11, Nov. 1995, pp. 68-74.
Jonathan Trevor, et al., "Exorcising Daemons: A Modular and Lightweight Approach to Deploying Applications on the Web," In Proceedings of the Fifth International World Wide Web Conference; Computer Networks and ISDN Systems, V. 28, Nos. 7-11, May 6-10, 1996, pp. 1053-1062. [Retrieved from http://www.ra.ethz.ch/CDstore/www5/www363/overview.htm 3/18/14].
Robert Thau, "Design Considerations for the Apache Server API," In Proceedings of the fifth International World Wide Web Conference; Computer Networks and ISDN Systems, V. 28, Nos. 7-11, May 6-10, 1996, 13 pages. [Retrieved from http://iw3c2.cs.ust.hk/WWW5/www5conf.inria.fr/fich_html/papers/P20/Overview.html 3/18/14].
Defendant Hewlett-Packard Company's Invalidity Contentions, *Implicit Networks, Inc., (Plaintiff)* v. *Hewlett-Packard Company,*

(56) **References Cited**

OTHER PUBLICATIONS

(Defendant), United States District Court Northern District of California San Francisco Division (Jun. 30, 2011), 27 pages. [Case No. 3:10-Cv-3746 Si].
Steve McCanne and Van Jacobson, "vic: A Flexible Framework for Packet Video," Proceedings of the third ACM international conference on Multimedia, ACM Multimedia 95—Electronic Proceedings, Nov. 5-9, 1995, 19 pages.
Dan Decasper et al., "Router Plugins A Software Architecture for next Generation Routers," Computer Communication Review, a publication of ACM SIGCOMM, vol. 28 No. 4, Oct. 1998, pp. 229-40.
David Mosberger, "Scout: a Path Based Operating System," Doctoral Dissertation Submitted to the University of Arizona, 1997, 174 pages.
Ion Stoica and Hui Zhang, "LIRA: An Approach for Service Differentiation in the Internet," Proceedings of 8th International Workshop on Network and Operating Systems Support for Digital Audio and Video, Jul. 8-10, 1998, 14 pages.
Oliver Spatscheck, "Defending Against Denial of Service Attacks in Scout," Proceedings of the 3rd Symposium on Operating Systems Design and Implementation, Feb. 1999, 15 pages.
Peter Deutsch et al., "GZIP file format specification version 4.3," Network Working Group, RFC 1952, May 1996, 24 pages. [Retrieved from http://tools.iettorg/html/rfc1952 3/18/14].
Kamran Husain and Jason Levitt, "Javascript Developer's Resource—client-side programming using HTML, netscape plug-ins and java applets," Prentice-Hall Inc., ISBN 0-13-267923-X, 1997, pp. 16, 141, 391-393, 416-418, 420, 428.
Netscape LiveWire Developer's Guide, Version 2.0 Part Number: UM151-02274-00, 1996, 190 pages.
Douglas Kramer, "The Java™ Platform," A White Paper, JavaSoft, May 1996, 25 pages.
Brian N. Bershad, et al., "Process for Rewriting Executable Content on a Network Server or Desktop Machine in Order to Enforce Site Specific Properties," U.S. Appl. No. 60/061,387, filed Oct. 7, 1997, pp. 1-8.
Emin Gun Sirer, et al., "Design and implementation of a distributed virtual machine for networked computers," 17th ACM Symposium on Operating System Principles (SOSP'99), Published as Operating Systems Review 34(5), Dec. 1999, pp. 202-216.
Emin Gun Sirer, Kimera: A System Architecture for Networked Computers, Department of Computer Science and Engineering, University of Washington, 1997, 2 pages.
Emin Gun Sirer, et al., Kimera Architecture, Department of Computer Science and Engineering, University of Washington, 1997, 4 pages.
Emin Gun Sirer, et al., "Distributed Virtual Machines: A System Architecture for Network Computing," Department of Computer Science and Engineering, University of Washington, Feb. 26, 1998, 4 pages.
Jonathan Aldrich, et al., "Static Analyses for Eliminating Unnecessary Synchronization from Java Programs," Department of Computer Science and Engineering, University of Washington, 1999, 20 pages.
Microsoft Active Server Pages: Frequently Asked Questions, Microsoft Corporation, Sep. 1997, MSI/IMPLICIT0002318, 4 pages.
Microsoft Internet Information Server—Web Server for Windows NT Operating System: Reviewer's Guide, Reviewing and Evaluating Microsoft Internet Information Server version 3.0, 1996, MS/IMPLICIT0004192, 45 pages.
Emin Gan Sirer, et al., "Improving the Security, Scalability, Manageability and Performance of System Services for Network Computing," Department of Computer Science and Engineering, University of Washington, 1998, 13 pages.
Exhibits 1-26, Defendant Microsoft Corporation's Invalidity Contentions Pursuant to Patent L.R. 3-3, *Implicit Networks, Inc.* (Plaintiff) vs. *Sybase, Inc. and Microsoft Corporation,* (Defendants), United States District Court Northern District of California San Francisco Division (Oct. 1, 2009), [Case No. 09-cv-01478].
Exhibits B1-B21, Defendant Microsoft Corporation's Invalidity Contentions Pursuant to Patent L.R. 3-3, *Implicit Networks, Inc.,* (Plaintiff) vs. *Sybase, Inc. and Microsoft Corporation,* (Defendants), United States District Court Northern District of California San Francisco Division (Oct. 1, 2009), [Case No. 09-cv-01478].
Begole, et al., "Transparent Sharing of Java Applets: A Replicated Approach," ACM UIST, 1997, pp. 55-64.
Newsome, et al.,"Proxy Compilation of Dynamically Loaded Java Classes with MoJo," ACM LCTES '02-Scopes '02, Jun. 19-21, 2002, pp. 204-212.
Benton, et al., "Compiling Standard ML to Java Bytecode," ACM ICFP, 1998, pp. 129-140.
Ahern, et al., "Formalising Java RMI with Explict Code Mobility," ACM OOPSLA, Oct. 16-20, 2005, pp. 403-422.
Kang, et al., "Query Type Classification for Web Documents Retrieval," ACM SIGIR, Jul. 28-Aug. 1, 2003, pp. 64-71.
Mukhtar, et al., "A Client Side Measurement Scheme for Request Routing in Virtual Open Content Delivery Networks," IEEE, 2003, pp. 235-242.
Olshefski, et al., "Understandiing the Management of Client Perceived Response Time," ACM SIGMetrics/Performance, Jun. 26-30, 2006, pp. 240-251.
Wirthlin, et al., "Web-Based IP Evaluation and Distribution Using Applets," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 22, No. 8, Aug. 2003, pp. 985-994.
Bonisch, et al., "Server Side 'Compresslets' for internet Multimedia Streams," IEEE, 1999, pp. 82-86.
Lai, et al., "On the Performance of Wide-Area Thin-Client Computing," ACM Transactions on Computer Systems, vol. 24, No. 2, May 2006, pp. 175-209.
Ding, et al., "Selective Java Applet Filtering on Internet," IEEE, 1999, pp. II-110-II-114.
Emin Gun Sirer, et al., "Distributed Virtual Machines: A System Architecture for Network Computing," Dept. of Computer Science & Engineering, University of Washington, Feb. 26, 1998, 4 pages.
Emin Gun Sirer, et al., "Design and Implementation of a Distributed Virtual Machine for Networked Computers," 17th ACM Symposium on Operating System Principles, published as Operating Systems Review, 34(5), Dec. 1999, pp. 202-216.

* cited by examiner

FIG. 1

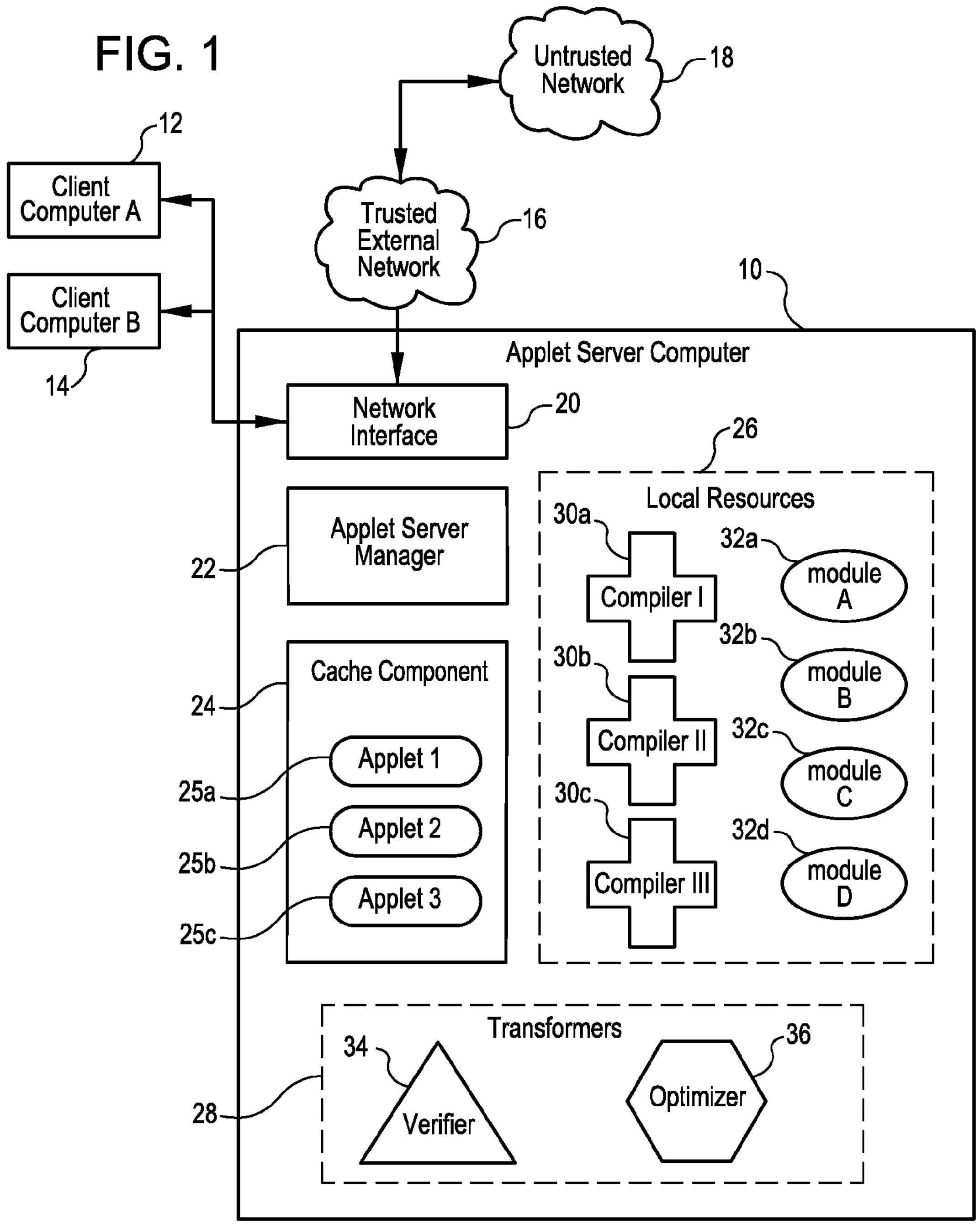

Untrusted Network
18
Client Computer A
12
Client Computer B
14
Trusted External Network
16
10
Applet Server Computer
Network Interface
20
26
Local Resources
Applet Server Manager
22
30a
Compiler I
32a
module A
32b
module B
30b
Compiler II
32c
module C
30c
Compiler III
32d
module D
Cache Component
24
Applet 1
25a
Applet 2
25b
Applet 3
25c
Transformers
34
Verifier
36
Optimizer
28

FIG. 2A

Request Data Type

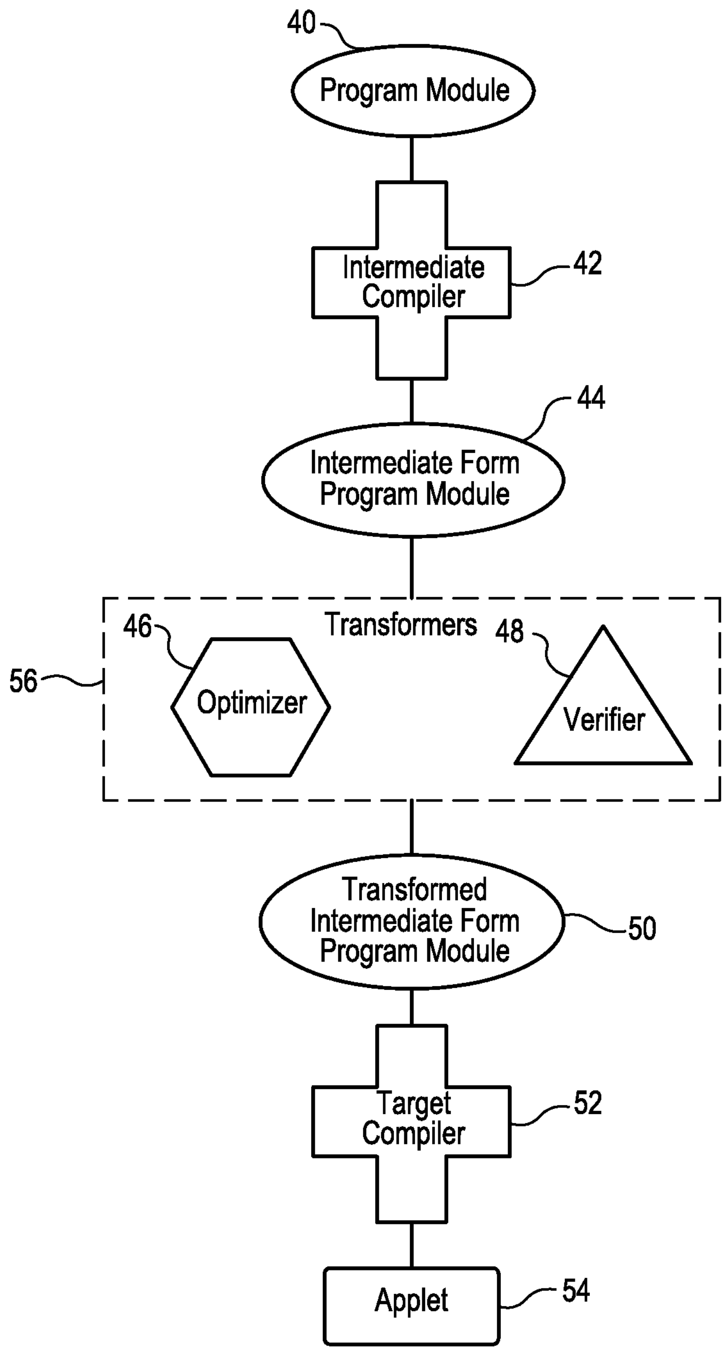
Tag	Value
Appplet - URL	(String) specifies the name of the requested applet.
Code -Type	(Source/Intermediate/Binary) specifies the format the applet is to be delivered to the requesting client in. A request for binary would specify the CPU of the requesting client (e.g., x 86)
Verification - Level	(0 - 100) specifies the degree of the verification to be performed. 0 = no/minimal verification, 100 = maximum verification (highest level of security).
Optimization - Level	(0 - 100) specifies the degree of optimization to be performed. 0 = no/minimal optimization, 100 = maximum optimization.

FIG. 2B

Code Data Type

Tag	Value
Appplet - URL	(String) specifies the name of the requested applet.
Code -Type	(Source/Intermediate/Binary) specifies the format the applet is to be delivered to the requesting client in. A request for binary would specify the CPU of the requesting client (e.g., x 86)
Verification - Level	(0 - 100) specifies the degree of the verification to be performed. 0 = no/minimal verification, 100 = maximum verification (highest level of security).
Optimization - Level	(0 - 100) specifies the degree of optimization to be performed. 0 = no/minimal optimization, 100 = maximum optimization.
Applet Length	(0 - 2^{32}) specifies the size of the requested applet.
Applet Code	The Requested Applet in the form specified by the requested data type.

FIG. 3


40
Program Module
Intermediate Compiler
42
44
Intermediate Form Program Module
Transformers
46
Optimizer
48
Verifier
56
Transformed Intermediate Form Program Module
50
Target Compiler
52
Applet
54

APPLICATION SERVER FOR DELIVERING APPLETS TO CLIENT COMPUTING DEVICES IN A DISTRIBUTED ENVIRONMENT

CROSS REFERENCES TO RELATED APPLICATIONS

This application is a continuation of U.S. patent application Ser. No. 11/933,161, filed Oct. 31, 2007 now U.S. Pat. No. 8,056,075, titled "Server Request Management", for all purposes including but not limited to the right of priority and benefit of earlier filing date, and expressly incorporates by reference the entire content of patent application Ser. No. 11/933,161 for all purposes. U.S. patent application Ser. No. 11/933,161 is a continuation of U.S. patent application Ser. No. 11/241,985 (now U.S. Pat. No. 7,774,740), filed Oct. 4, 2005, titled "Application Server". U.S. patent application Ser. No. 11/241,985 is a continuation of U.S. patent application Ser. No. 09/968,704 (now U.S. Pat. No. 6,976,248), filed Oct. 1, 2001, titled "Application Server Facilitating with Client's Computer for Applets Along with Various Formats". U.S. patent application Ser. No. 09/968,704 is a continuation of U.S. patent application Ser. No. 09/040,972 (now U.S. Pat. No. 6,324,685), filed Mar. 18, 1998, titled "Application Server". This application claims the benefit of the following applications for all purposes including but not limited to the right of priority and benefit of earlier filing date, and expressly incorporates by reference the entire content of the following applications for all purposes: U.S. patent application Ser. No. 11/933,161; U.S. patent application Ser. No. 11/241,985; U.S. patent application Ser. No. 09/968,704; and U.S. patent application Ser. No. 09/040,972.

FIELD OF THE INVENTION

The present invention relates to computer operating system and, in particular to a server architecture providing application caching and security verification.

BACKGROUND OF THE INVENTION

The following application is incorporated by reference as if fully set forth herein: U.S. application Ser. No. 11/241,985 filed Oct. 4, 2005.

The growth of the Internet's importance to business, along with the increased dependence upon corporate networks, has created a demand for more secure and efficient computer systems. The traditional solution to this problem has been to depend upon improvements in hardware performance to make up for the performance penalty that is typically incurred when a computer system is made more secure and stable. Increased interconnectivity has also created a need for improved interoperability amongst a variety of computers that are now connected to one another. One solution to the problem of the variety of computers interconnected via the Internet and corporate networks has been the development of portable architecture neutral programming languages. The most widely known of these is the Java™ programming language, though, there are numerous other-architecture neutral languages.

Architecture neutral programming languages allow programs downloaded from a server computer to a client computer to be interpreted and executed locally. This is possible because the compiler generates partially compiled intermediate byte-code, rather than fully compiled native machine code. In order to run a program, the client machine uses an interpreter to execute the compiled byte-code. The byte-codes provide an architecture neutral object file format, which allows the code to be transported to multiple platforms. This allows the program to be run on any system which implements the appropriate interpreter and run-time system. Collectively, the interpreter and runtime system implement a virtual machine. This structure results in a very secure language.

The security of this system is premised on the ability of the byte-code to be verified independently by the client computer. Using the Java™ programming language or some other virtual machine implementing technology, a client can ensure that the downloaded program will not crash the user's computer or perform operations for which it does not have permission.

The traditional implementations of architecture neutral languages are not without problems. While providing tremendous cross platform support, the current implementations of architecture neutral languages require that every client performs its own verification and interpretation of the intermediate code. The high computation and memory requirements of a verifier, compiler and interpreter restrict the applicability of these technologies to powerful client computers.

Another problem with performing the verification process on the client computer is that any individual within an organization may disable some or of the checks performed on downloaded code. The current structure of these systems makes security management at the enterprise level almost impossible. Since upgrades of security checking software must be made on every client computer, the cost and time involved in doing such upgrades makes it likely that outdated or corrupt copies of the verifier or interpreter exist within an organization. Even when an organization is diligent in maintaining a client based security model, the size of the undertaking in a large organization increases the likelihood that there will be problems.

There is a need for a scalable distributed system architecture that provides a mechanism for client computers to request and execute applets in a safe manner without requiring the client machines to have local resources to compile or verify the code. There is a further need for a system in which the applets may be cached in either an intermediate architecture neutral form or machine specific form in order to increase overall system performance and efficiency.

SUMMARY OF THE INVENTION

in accordance with one embodiment of the invention, an applet server architecture is taught which allows client computers to request and execute applets in a safe manner without requiring the client to have local resources to verify or compile the applet code. Compilation and byte code verification in the present invention are seer based and thereby provide more efficient use of resources and a flexible mechanism for instituting enterprise-wide security policies. The server architecture also provides a cache for applets, allowing clients to receive applet code without having to access nodes outside the local network. The cache also provides a mechanism for avoiding repeated verification and compilation of previously requested applet code since any client requesting a given applet will have the request satisfied by a single cache entry.

Machine specific binary code is essentially interpreted code since the processor for a given computer can essentially be viewed as a form of an interpreter, interpreting binary code into the associated electronic equivalents. The present invention adds a level of indirection in the form of an intermediate language that is processor independent. The intermediate language serves as the basis for security verification, code optimizations, or any other compile time modifications that might be necessary. The intermediate form allows a single version of the source to be stored for many target platforms instead of having a different binary fir each potential target computer. Compilations to the target form can either be done at the lime of a cache hit or they can be avoided all together if the target machine is able to directly interpret the intermediate form. If the compilation is done on the server, then a copy of the of the compiled code as well as the intermediate form can be stored in the cache. The performance advantage derived from caching the compiled form as well as the intermediate depends upon the number of clients with the same CPU.

The novel features believed characteristic of the invention are set forth in the appended claims. The invention itself, however, as well as other features and advantages thereof will best be understood by reference to the detailed description which follows, when read in conjunction with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is diagram showing the major components which may be used to implement an applet server in one embodiment of the present invention;

FIG. **2***a* is a table which illustrates the structure of the request format data type;

FIG. **2***b* is a table which illustrates the structure of the returned code data type.

FIG. **3** is a diagram showing the compilation and transformation of a program module into an applet in a particular form.

DETAILED DESCRIPTION OF ME INVENTION

Referring to FIG. **1**, an applet server architecture according to one embodiment of the invention is based on an applet server computer **10** which in turn is connected to client computer A **12**, client computer B **14**, an external network **16** and an untrusted network **18**. The applet server computer **10** connects to client computers **12** and **14**, an external network **16**, and an untrusted network **18** by means of a network interface **20**. Typically this connection will involve one or more of the computers or networks having a connection to the Internet.

The applet server computer **10** accomplishes its objectives by manipulating computer programs in several formats. An applet (e.g. applets **1**-**3**, **25***a*-**25***c*) is any form of program instructions, whether in binary, source or intermediate format. In the case of this architecture, the applet code can either be a self-contained program, or it can be a code fragment associated with a larger application.

Binary format refers to processor specific machine instructions suitable for running natively on a given computing platform (also referred to as "target" because of the concept of "targeting" a compiler to produce binary code for a given processor type).

Source refers to non-binary applet code, generally in the form of higher level languages (i.e. the C™, C++™, Java™, Visual Basic™, ActiveX™, Fortran™, and Modula™ programming languages).

Intermediate format refers to a common intermediate byte-code that is produced by compiling a given source code put. The intermediate byte-code need not necessarily be Java™ byte-code.

Treating applets in this general sense allows client computers **12** and **14** to request not only applications, but portions of applications. Client computers **12** and **14** are thus able to use applet server computer **10** as the equivalent of a loader, loading in appropriate parts of the application in the form of applets. In turn client computers **12** and **14** can run large applications without requiring that the client computers **12** and **14** have the resources to store the entire application in memory at once.

Having the applets delivered from applet server computer **10** allows code in intermediate form to be verified, optimized, and compiled before being transmitted to client computers **12** and **14**. This reduces the amount of work the client computers **12** and **14** have to do and provides a convenient way to impose global restrictions on code.

In operation, client computer A **12** transmits a request to an applet server computer **10** requesting an applet in a particular form. The form may be selected from a large matrix of many possible for that can be recognized by the system. The request specifies the format (source, intermediate, or binary) which the client wishes to receive the applet. The request may also specify that the applet be verified or have some other transformation operation preformed upon it. Verification, optimization and compression are examples of types of transformation operations. The request is received by the network interface **20** of the applet server computer **10** which passes the request onto the applet server manager **22**.

After interpreting the request, the applet server manager **22** checks to see if the requested applet is available in the cache **24**. The cache **24** stores applets in a variety of formats (source, intermediate, or binary). If the requested form of the applet is available in the cache **24** (applet 1 **25***a*, applet 2 **25***b*, or applet 3 **25***c* in this example) the applet server manager **22** instructs the network interface **20** to transmit the applet to requesting client computer A **12**.

If the requested applet is not available in the cache **24**, then the apple server manager **22** will attempt to build the requested applet from local resources **26** and one or more transformation operations performed by one or more of the transformers **28**. Local resources **26** are comprised of compilers **30***a*, **30***b* and **30***c* and program code modules **32***a*, **32***b*, **32***c* and **32***d*. The requested applet is built by selecting one or more program code modules **32** and compiling them with one or more compilers **30**. Transformer operations may be performed by the verifier **34** or the optimizer **36**. After the applet server manager **22** builds the applet, the network interface **20** transmits the applet to the requesting client computer A **12**.

If the request can not be satisfied by building the applet from local resources **26** and transformers **28**, the applet server manager **22** will pass a request for the requested applet to external network **16** and/or untrusted network **18**. The applet server manager **22** may request the applet in intermediate form or in executable form or it may request the local resources **26** and transformers **28** it needs to complete building the applet itself.

The cache **24** is capable of responding to the following commands: GET, PUT, and FLUSH. GET is used to retrieve a given applet from the cache. PUT is used to store an applet in the cache. FLUSH. is used to clear the cache of one or more entries. When the cache is unable to locate an item in response to a GET operation, it returns a cache miss. The program which issued the GET command is then responsible for locating the desired form of the applet by other means and optionally storing it in the cache when it is retrieved (using the PUT operation). The FLUSH command will clear the cache of one or more entries and any subsequent GETs for the FLUSHed applet code will result in a cache miss. This is useful if a particular applet needs to be updated from a remote server on a periodic basis. When using PUT, the program issuing the command specifies a time to live (TTL) in the cache. When the TTL expires, the cache entry is removed by means of a FLUSH operation.

Local resources **26** are comprised of program modules **32** (applets in source form, not the requested form) and compilers **30**. The program modules **32** are run through the compilers **30** in order to produce applets in the requested form. The applet server manager **20** may also direct the modules **32** to be processed by a verifier **34** or another transformer such as an optimizer **36**. Program modules **32** are program code used to build applets. Program modules **32** may be stored in local resources **26** in source, binary, or intermediate formats. When an applet is built it may require the operation of one or more compilers **30** upon one or more program modules **32**. The program modules **32** may be combined and recompiled with previously cached applets and the resulting applet may also be cached for use at a future time. Additionally, program modules **32**, compilers **30** and transformers **28** (including verifiers **34** and optimizers **36**) may be distributed across a network. The applet server manager **22** may pass requests for the components it needs to build a particular applet back to the network interface **20** which in turn passes the request onto the rest of the network and may include external network **16** and untrusted network **18**.

FIG. **3** provides further illustration of how an applet is produced from local resources and transformers. In this illustration the request is for an optimized and verified applet compiled to a machine specific form. A program module **40** is compiled into an intermediate form program module **44** by an intermediate compiler **42**. The intermediate form program module **44** is then transformed by an optimizer **46** or a verifier **48**. The resulting transformed intermediate form program module **50** is then compiled by target compiler **52** into machine specific code applet **54**.

There are two types of compilers used to build applets: intermediate compilers **42** and target compilers **52**. The intermediate compiler **42** compiles program modules (source applet code) **40** and produces a common intermediate pseudo-binary representation of the source applet code (intermediate form program module **44**). The word pseudo is used because the intermediate form **44** is not processor specific but is still a binary representation of the source program module **40**. This intermediate form can be re-targeted and compiled for a particular processor. Alternatively, the intermediate form **44** can be interpreted by an interpreter or virtual machine that understands the internal binary representation of the intermediate form. A target compiler **52** compiles intermediate applet code **44** into an applet **54** in a processor specific format (binary) suitable for running natively on a given computing platform.

Transformers **56** are programs that take in intermediate code and put out intermediate code. Transformers **56** are generally used for things like verification and optimization. Other transformers might include compressors that identify portions of code that can be replaced with smaller equivalents. Transformers can be matched up to any other component that takes in intermediate code as an input. These include the cache **24** and the target compilers **52**. Global policies for transformers **56** can be implemented which ensure that all applets are run through some set of transformers before being returned to the client.

A verifier **48** is a type of transformer that is able to analyze input code and determine areas that might not be safe. The verifier **48** can determine the level of safety. Some verifiers **48** look for areas where unsafe or protected memory is being accessed, others might look for accesses to system resources such as IO devices. Once a verifier **48** determines the portion

of unsafe applet code several steps can be taken. The offending code portion can be encased with new code that specifically prevents this unsafe code section from being executed. The unsafe code can be modified to be safe. The unsafe code can be flagged in such a way that a user can be warned about the possible risk of executing the code fragment. The verifier's role can therefore be summarized as determining where unsafe code exists and possibly altering the offending code to render it harmless. Verifiers **48** can operate on any format of input code, whether in source, intermediate or binary form. However, since intermediate code is a common format, it is most efficient to have a single verifier that will operate on code in this format. This eliminates the need to build specific knowledge of various source languages into the verifier. Verifiers **48** are a form of a transformer. Verifiers **48** take in intermediate code and put out verified intermediate code. Verifiers **48** are responsible for identifying non-secure portions of code in the intermediate code and modifying this code to make it secure. Security problems generally include access to memory areas that are unsafe (such as system memory, or memory outside the application space of the applet).

The choice of adding in the verification step can be left up to the client computer **12**, the applet server computer **10** (see FIG. **1**), or can be based on the network that the applet originated from. Server managers can institute global policies that affect all clients by forcing all applets to be run through the verifier **48**. Alternatively, verification can be reserved for un trusted networks (**18** in FIG. **1**), or it can be left up to the client to determine whether the verification should be performed. In the preferred embodiment, verification levels determined by the applet server **10**. In this way, a uniform security policy may be implemented from a single machine (i.e., the applet server **10**).

Optimizers **46** are another type of transformer program. Optimizers **46** analyze code, making improvements to well known code fragments by substituting in optimized but equivalent code fragments. Optimizers **46** take in intermediate code **44** and put out transformed intermediate code **50**. The transformed intermediate code **50** is functionally equivalent to the source intermediate code **44** in that they share the same structure.

Referring again to FIG. **1**, policies may be instituted on the applet server **10** that force a certain set of request parameters regardless of what the client asked for. For example, the applet server manager **22** can run the applet through a verifier **34** or optimizer **36** regardless of whether the client **12** requested this or not. Since the server **10** might have to go to an untrusted network **18** to retrieve a given applet, it evil then run this applet through the required transformers **28**, particularly the verifier **34** before returning it to the client **12**. Since clients **12** and **14** have to go through the applet server computer **10**, this ensures that clients **12** and **14** do not receive applets directly from an untrusted network **18**. In addition, since the server will be dealing directly with untrusted network **18**, it can be set up to institute policies based on the network. A trusted external network **16** may be treated differently than an untrusted network **18**. This will provide the ability to run a verifier **34** only when dealing with an entrusted network **18**, but not when dealing with a trusted external network **16**. In one embodiment, intermediate code is passed through a verifier **34** and the source of the code merely determines the level of verification applied.

The client **12** is the target computer on which the user wishes to execute an applet. The client **12** requests applets from the server **10** in a specific form. Applets can be requested in various formats including source, intermediate and binary. In addition, an applet can be requested with verification and/

or other compile time operations. Optionally, the client **12** can pass a verifier to the server to provide verification. If the server **10** implements its own security, then both the client and server verifiers will be run. The verifier that is passed from the client to the server is cached at the server fix subsequent verification. The client can refer to this verifier by a server-generated handle to avoid having to pass the verifier each time an applet is requested,

Client computers **12** and **14** requesting applet code in intermediate format need to have an interpreter or virtual machine capable of interpreting the binary code in the intermediate format if the applet is to be executed on the client machine.

In the preferred embodiment, requests to the applet server are in a format similar to those of an HTTP header and are comprised of tags and values. In one embodiment, an HTTP GET method is used to make the request (though use of the HTTP protocol is not necessary to implement the present invention). The request is made up of a series of tags which specify the requested applet, the platform on which it is to be run and the type of code (source/intermediate/binary), a verification level and an optimization level. New tags and values can be added to extend functionality as needed and the applet server manager **22** will discard any tag it does not recognize. When the applet server computer **10** returns the requested applet to the requesting client computer A **12**, it will transmit the request header followed by the applet code. In this instance, the header will additionally include a field which defines the length of the applet code. FIG. **2** provides a table which illustrates the request format and the returned code format.

While this invention has been described with reference to specific embodiments, this description is not meant to limit the scope of the invention. Various modifications of the disclosed embodiments, as well as other embodiments of the invention, will be apparent to persons skilled in the art upon reference to this description. It is therefore contemplated that the appended claims will cover any such modifications or embodiments as fall within the scope of the invention.

The invention claimed is:

1. A method, comprising:

receiving an applet request at a computer system from a first client computer, wherein the applet request specifies a particular applet for the first client computer and specifies one or more client parameters for the first client computer;

the computer system processing the applet request to cause, the particular applet to be generated, wherein the generated particular applet includes source code that is in a form based on the specified one or more client parameters for the first client computer; and

sending the particular applet from the computer system to the first client computer.

2. The method of claim **1**, wherein the source code is in a high-level programming language.

3. The method of claim **1**, wherein the form of the source code is suitable for running on the first client computer.

4. The method of claim **1**, further comprising:

the computer system storing the particular applet in an applet cache;

receiving a second applet request for the particular applet, wherein the second applet request is for a second client computer and specifies one or more client parameters for the second client computer; and

sending the particular applet from the computer system to the second client computer from the applet cache without re-generating the particular applet.

5. The method of claim **4**, wherein sending the particular applet to the second client computer is in response to a determination that, based on at least one of the one or more client parameters for the first client computer and at least one of the one or more client parameters for the second client computer, the cached particular applet is suitable for the second client computer.

6. The method of claim **1**, further comprising:

prior to causing the particular applet to be generated in response to the applet request, checking to see if the particular applet is present in an applet cache accessible to the computer system.

7. The method of claim **1**, wherein causing the at least one executable component to be generated includes sending a request to a different computer system located in a network that is external to the computer system.

8. The method of claim **1**, wherein causing the applet to be generated includes the computer system:

sending a request to a different computer system located in a network that is external to the computer system;

receiving a source code module from the different computer system; and

including the received source code module in the particular applet.

9. The method of claim **1**, wherein causing the applet to be generated includes the computer system transforming the received source code module to generate the source code included in the particular applet.

10. The method of claim **1**, wherein the one or more client parameters for the first client computer include a requested security parameter.

11. The method of claim **1**, wherein the applet request is received via HTTP in association with a web page, and wherein the particular applet is a web applet.

12. A non-transitory computer-readable storage medium having stored thereon instructions that are executable to cause a computer system to perform operations comprising:

receiving an applet request from a first client computer for a particular applet, wherein the applet request specifies one or more parameters for the particular applet that are based on one or more characteristics of the client computer;

acquiring a specific form of the particular applet that includes source code, based on the specified one or more parameters in the applet request, wherein the specific form complies with the specified one or more parameters; and

sending the specific form of the particular applet to the first client computer in response to the applet request.

13. The non-transitory computer-readable storage medium of claim **12**, wherein acquiring the specific form of the particular applet includes acquiring one or more applet components from an external trusted network.

14. The non-transitory computer-readable storage medium of claim **12**, wherein acquiring the specific form of the particular applet includes acquiring one or more applet components from an external untrusted network; and

wherein the operations further comprise verifying the one or more applet components prior to sending the specific form of the particular applet to the first client computer.

15. The non-transitory computer-readable storage medium of claim **12**, wherein acquiring the specific form of the particular applet includes:

in response to failing to locate the specific form of the particular applet within the computer system, receiving at least a portion of the applet from another computer system; and generating the specific form of the particular applet prior to sending the specific form of the particular applet to the first client computer.

16. The non-transitory computer-readable storage medium of claim **12**, wherein the operations further comprise performing a first transformation operation on at least one applet component of the particular applet, and performing a second transformation operation on at least a different applet component of the particular applet.

17. The non-transitory computer-readable storage medium of claim **16**, wherein the first transformation operation is a compression operation and the second transformation operation is a verification operation.

18. A computer system, comprising:

a processor; and

a non-transitory computer-readable storage medium having stored thereon instructions that are executable by the processor to cause the computer system to perform operations comprising:

receiving an applet request from a first client computer, wherein the applet request specifies a particular applet for the first client computer and specifies one or more client parameters for the first client computer;

processing the applet request, including causing at least a portion of the particular applet to be generated, based on the specified one or more client parameters for the first client computer, in response to the applet request, wherein the generated applet includes source code in a form that is based on the one or more client parameters; and

causing the particular applet to be sent to the first client computer.

19. The computer system of claim **18**, wherein the operations further comprise:

caching the particular applet for future requests from other client computers.

20. The computer system of claim **18**, wherein the operations further comprise:

optimizing the particular applet for the first client computer based on the one or more client parameters for the first client computer.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO.	: 8,856,779 B2	Page 1 of 1
APPLICATION NO.	: 13/269905	
DATED	: October 7, 2014	
INVENTOR(S)	: Edward Balassanian	

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

In the Claims

In column 7, line 48 (claim 1), please delete "cause," and insert --cause--.

In column 8, line 41 (claim 12), please delete "client computer" and insert --first client computer--.

Signed and Sealed this
Sixth Day of January, 2015

Michelle K. Lee

Michelle K. Lee
Deputy Director of the United States Patent and Trademark Office

EXHIBIT 3

US009325740B2

(12) **United States Patent**
Balassanian

(10) **Patent No.: US 9,325,740 B2**
(45) **Date of Patent: *Apr. 26, 2016**

(54) **APPLICATION SERVER FOR DELIVERING APPLETS TO CLIENT COMPUTING DEVICES IN A DISTRIBUTED ENVIRONMENT**

(71) Applicant: **IMPLICIT, LLC**, Seattle, WA (US)

(72) Inventor: **Edward Balassanian**, Seattle, WA (US)

(73) Assignee: **Implicit, LLC**, Seattle, WA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **14/507,394**

(22) Filed: **Oct. 6, 2014**

(65) **Prior Publication Data**

US 2015/0089581 A1 Mar. 26, 2015

Related U.S. Application Data

(63) Continuation of application No. 13/269,905, filed on Oct. 10, 2011, now Pat. No. 8,856,779, which is a continuation of application No. 11/933,161, filed on Oct. 31, 2007, now Pat. No. 8,056,075, which is a continuation of application No. 11/241,985, filed on Oct. 4, 2005, now Pat. No. 7,774,740, which is a continuation of application No. 09/968,704, filed on Oct. 1, 2001, now Pat. No. 6,976,248, which is a continuation of application No. 09/040,972, filed on Mar. 18, 1998, now Pat. No. 6,324,685.

(51) **Int. Cl.**

G06F 9/445	(2006.01)
H04L 29/06	(2006.01)
H04L 29/08	(2006.01)

(52) **U.S. Cl.**
CPC ***H04L 63/20*** (2013.01); ***G06F 9/445*** (2013.01); ***G06F 9/44526*** (2013.01); ***G06F 9/44589*** (2013.01); ***H04L 67/02*** (2013.01); ***H04L 67/10*** (2013.01); ***H04L 67/2842*** (2013.01); ***H04L 67/34*** (2013.01)

(58) **Field of Classification Search**
USPC 717/168–177; 709/203
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,706,502	A	1/1998	Foley et al.
5,761,421	A	6/1998	van Hoff et al.

(Continued)

OTHER PUBLICATIONS

Heines, "Enabling XML Storage from Java Applets in a GUI Programming Course", ACM, The SIGCSE Bulletin, vol. 35, No. 2, pp. 88-93, 2003.*

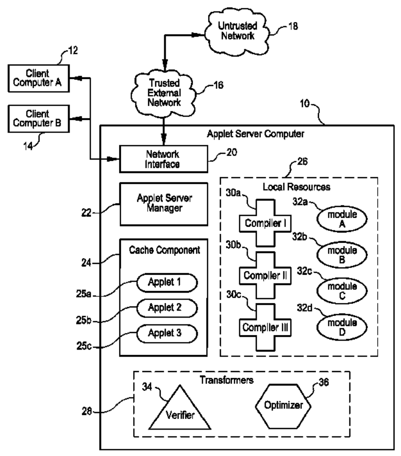
(Continued)

Primary Examiner — Anil Khatri
(74) *Attorney, Agent, or Firm* — Meyertons, Hood, Kivlin, Kowert & Goetzel, P.C.

(57) **ABSTRACT**

An applet server accepts requests for applets from client computers. A request specifies the format in which an applet is to be delivered to the requesting client computer. The applet server has a cache used to store applets for distribution to client computers. If the specified form of the requested applet is available in the cache, the applet server transmits the applet to the requesting client. If the applet is not available in the cache, the server will attempt to build the applet from local resources (program code modules and compilers) and transformer programs (verifiers and optimizers). If the applet server is able to build the requested applet, it will transmit the applet to the requesting client computer. If the applet server is unable to build the requested applet, it will pass the request to another applet server on the network for fulfillment of the request.

20 Claims, 3 Drawing Sheets



(56) References Cited

U.S. PATENT DOCUMENTS

5,805,829	A	9/1998	Cohen et al.	
5,828,840	A	10/1998	Cowan et al.	
5,835,712	A	11/1998	DuFresne	
5,848,274	A	12/1998	Hamby et al.	
5,872,915	A	2/1999	Dykes et al.	
5,884,078	A	3/1999	Faustini	
5,911,776	A	6/1999	Guck	
5,920,725	A	7/1999	Ma et al.	
5,923,885	A	7/1999	Johnson et al.	
5,926,631	A	7/1999	McGarvey	
5,943,496	A	8/1999	Li et al.	
5,944,784	A	8/1999	Simonoff et al.	
5,996,022	A	11/1999	Krueger et al.	
6,105,063	A	8/2000	Hayes, Jr.	
6,195,794	B1	2/2001	Buxton	
6,212,673	B1	4/2001	House et al.	
6,230,184	B1	5/2001	White et al.	
6,253,228	B1	6/2001	Ferris et al.	
6,279,151	B1	8/2001	Breslau et al.	
6,282,702	B1	8/2001	Ungar	
6,295,643	B1	9/2001	Brown et al.	
6,317,781	B1	11/2001	De Boor et al.	
6,321,377	B1	11/2001	Beadle et al.	
6,324,685	B1	11/2001	Balassanian	
6,327,701	B2	12/2001	Ungar	
6,330,710	B1	12/2001	O'Neil et al.	
6,367,077	B1 *	4/2002	Brodersen et al.	717/170
6,381,742	B2 *	4/2002	Forbes et al.	717/176
6,434,745	B1	8/2002	Conley et al.	
6,446,081	B1	9/2002	Preston	
6,502,236	B1	12/2002	Allen et al.	
6,546,554	B1	4/2003	Schmidt et al.	
6,594,820	B1	7/2003	Ungar	
6,611,858	B1	8/2003	Aravamudan et al.	
6,636,900	B2	10/2003	Abdelnur	
6,643,683	B1	11/2003	Drumm et al.	
6,704,926	B1	3/2004	Blandy et al.	
6,718,364	B2	4/2004	Connelly et al.	
6,718,540	B1	4/2004	Azua et al.	
6,741,608	B1	5/2004	Bouis et al.	
6,742,165	B2	5/2004	Lev et al.	
6,745,386	B1	6/2004	Yellin	
6,754,693	B1	6/2004	Roberts et al.	
6,757,894	B2	6/2004	Eylon et al.	
6,766,366	B1	7/2004	Schafer et al.	
6,772,408	B1	8/2004	Velonis et al.	
6,789,252	B1	9/2004	Burke et al.	
6,802,061	B1	10/2004	Partovi et al.	
6,832,263	B2	12/2004	Polizzi et al.	
6,836,889	B1	12/2004	Chan et al.	
6,842,897	B1	1/2005	Beadle et al.	
6,865,732	B1	3/2005	Morgan	
6,865,735	B1	3/2005	Sirer et al.	
6,910,128	B1	6/2005	Skibbie et al.	
6,947,943	B2	9/2005	DeAnna et al.	
6,950,850	B1	9/2005	Leff et al.	
6,976,248	B2	12/2005	Balassanian	
6,990,513	B2 *	1/2006	Belfiore	G06F 9/54 707/E17.005
6,993,743	B2	1/2006	Crupi et al.	
6,996,817	B2 *	2/2006	Birum et al.	717/170
7,051,315	B2	5/2006	Artzi et al.	
7,069,294	B2 *	6/2006	Clough et al.	709/203
7,127,700	B2	10/2006	Large	
7,131,111	B2	10/2006	Passanisi	
7,131,122	B1	10/2006	Lakhdhir	
7,136,896	B1	11/2006	Srinivas et al.	
7,150,015	B2	12/2006	Pace et al.	
7,155,715	B1	12/2006	Cui et al.	
7,281,245	B2 *	10/2007	Reynar et al.	717/173
7,346,655	B2	3/2008	Donoho et al.	
7,415,706	B1	8/2008	Raju et al.	
7,434,215	B2	10/2008	Boykin et al.	
7,444,629	B2	10/2008	Chirakansakcharoen et al.	
7,472,171	B2	12/2008	Miller et al.	
7,493,591	B2	2/2009	Charisius et al.	
7,519,684	B2	4/2009	Backhouse et al.	
7,523,158	B1	4/2009	Nickerson et al.	
7,530,050	B2	5/2009	Mohan et al.	
7,536,686	B2 *	5/2009	Tan	G06F 8/61 713/100
7,562,346	B2	7/2009	Jhanwar et al.	
7,562,358	B2 *	7/2009	Bennett	G06F 9/5055 709/220
7,590,643	B2	9/2009	Demiroski et al.	
7,614,052	B2 *	11/2009	Wei	717/176
7,624,394	B1	11/2009	Christopher, Jr.	
7,665,082	B2 *	2/2010	Wyatt et al.	717/171
7,703,093	B2	4/2010	Fischer et al.	
7,707,571	B1	4/2010	Harris et al.	
7,721,283	B2	5/2010	Kovachka-Dimitrova et al.	
7,730,482	B2	6/2010	Illowsky et al.	
7,774,742	B2	8/2010	Gupta et al.	
7,814,142	B2 *	10/2010	Mamou	G06F 17/30563 709/203
7,814,475	B2 *	10/2010	Cohen et al.	717/168
7,934,212	B2	4/2011	Lakhdhir	
7,984,121	B2 *	7/2011	Konopka et al.	709/220
7,991,834	B2	8/2011	Ferris et al.	
8,056,075	B2	11/2011	Balassanian	
8,127,274	B2	2/2012	Astheimer	
8,146,077	B2 *	3/2012	McNally	G06F 17/30905 717/174
8,285,777	B2	10/2012	Giles et al.	
8,392,906	B2	3/2013	Broussard et al.	
8,392,912	B2	3/2013	Davis et al.	
8,418,170	B2 *	4/2013	Saxton	G06F 8/61 717/174
8,490,082	B2	7/2013	Moore et al.	
8,499,278	B2	7/2013	Hughes	
8,612,966	B2 *	12/2013	Huff et al.	717/174
8,615,545	B1	12/2013	Lakhdhir	
8,762,988	B2 *	6/2014	Kong et al.	717/177
8,826,266	B2 *	9/2014	Little	G06F 9/541 717/168
8,832,679	B2 *	9/2014	Suchy	G06F 8/61 717/100
8,863,114	B2 *	10/2014	Shah	G06F 8/60 717/175
8,954,952	B2 *	2/2015	Guizar	G06F 8/60 717/168

OTHER PUBLICATIONS

Schlumberger et al, "Jarhead Analysis and Detection of Malicious Java Applets", ACM, pp. 249-258, 2012.*

Yang et al, "Developing Integrated Web and Database Applications a Using JAVA Applets and JDBC Drivers", ACM, pp. 302-306, 1998.*

Barbuti et al, "Java Bytecode Verification on Java Cards", ACM, pp. 431-438, 2004.*

Ding et al, "Selective Java Applet Filtering on Internet", IEEE, pp. 110-114, 1999.*

Shishir Gundavaram, CGI Programming on the World Wide Web, Mar. 1996, Chapters 1-12, 323 pages [Retrieved from http://docstore.mik.ua/orelly/web/cgi/ Jan. 28, 2015].

Chuck Musciano and Bill Kennedy, HTML: The Definitive Guide, May 1997, Chapters 1-15, 454 pages [Retrieved from http://docstore.mik.ua/orelly/web/html/ Jan. 28, 2015].

David Flannagan, JavaScript: The Definitive Guide, Jan. 1997, Chapters 1-20, 343 pages [Retrieved from http://docstore.mik.ua/orelly/web/jscript/ Jan. 28, 2015].

Wall, et al., Programming Perl, Sep. 1996, Chapters 1-9, 516 pages [Retrieved from http://docstore.mik.ua/orelly/web/perl/ Jan. 28, 2015].

Stephen Spainhour and Valerie Quercia, Oct. 1996, Chapters 1-26, 403 pages [Retrieved from http://docstore.mik.ua/orelly/web/webnut/ Jan. 28, 2015].

Exhibits 1-26, Defendant Microsoft Corporation's Invalidity Contentions Pursuant to Patent L.R. 3-3, *Implicit Networks, Inc.* (*Plaintiff*) vs. *Sybase, Inc. and Microsoft Corporation*, (*Defendants*), United States District Court Northern District of California San Francisco Division (Oct. 1, 2009), [Case No. 09-cv-01478].

(56) **References Cited**

OTHER PUBLICATIONS

Exhibits B1-B21, Defendant Microsoft Corporation's Invalidity Contentions Pursuant to Patent L.R. 3-3, *Implicit Networks, Inc.*, (*Plaintiff*) vs. *Sybase, Inc. and Microsoft Corporation*, (*Defendants*), United States District Court Northern District of California San Francisco Division (Oct. 1, 2009), [Case No. 09-cv-01478].
Begole, et al., "Transparent Sharing of Java Applets: A Replicated Approach," ACM UIST, 1997, pp. 55-64.
Newsome, et al., "Proxy Compilation of Dynamically Loaded Java Classes with MoJo," ACM LCTES '02-SCOPES '02, Jun. 19-21, 2002, pp. 204-212.
Benton, et al., "Compiling Standard ML to Java Bytecode," ACM ICFP, 1998, pp. 129-140.
Ahern, et al., "Formalising Java RMI with Explicit Code Mobility," ACM OOPSLA, Oct. 16-20, 2005, pp. 403-422.
Kang, et al., "Query Type Classification for Web Documents Retrieval," ACM SIGIR, Jul. 28-Aug. 1, 2003, pp. 64-71.
Mukhtar, et al., "A Client Side Measurement Scheme for Request Routing in Virtual Open Content Delivery Networks." IEEE, 2003, pp. 235-242.
Olshefski, et al., "Understanding the Management of Client Perceived Response Time," ACM SIGMetrics/Performance, Jun. 26-30, 2006, pp. 240-251.
Wirthlin, et al., "Web-Based IP Evaluation and Distribution Using Applets," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 22, No. 8, Aug. 2003, pp. 985-994.
Bonisch, et al., "Server Side 'Compresslets' for Internet Multimedia Streams," IEEE, 1999, pp. 82-86.
Lai, et al., "On the Performance of Wide-Area Thin-Client Computing," ACM Transactions on Computer Systems, vol. 24, No. 2, May 2006, pp. 175-209.
Ding, et al., "Selective Java Applet Filtering on Internet," IEEE, 1999, pp. II-110-II-114.
Emin Gun Sirer, et al., "Distributed Virtual Machines: A System Architecture for Network Computing," Dept. of Computer Science & Engineering, University of Washington, Feb. 26, 1998, 4 pages.
Emin Gun Sirer, et al., "Design and Implementation of a Distributed Virtual Machine for Networked Computers," 17th ACM Symposium on Operating System Principles, published as Operating Systems Review, 34(5), Dec. 1999, pp. 202-216.
Defendant Microsoft Corporation's Invalidity Contentions Pursuant to Patent L.R. 3-3, *Implicit Networks, Inc.*, (*Plaintiff*) vs. *Sybase, Inc. and Microsoft Corporation*, (*Defendants*), United States District Court Northern District of California San Francisco Division (Oct. 1, 2009), including Exhibits A & B, 12 pages. [Case No. 09-cv-01478].
Eric A. Meyer and Peter Murray, "Borealis Image Server," In Proceedings of the Fifth International World Wide Web Conference; Computer Networks and ISDN Systems, V. 28, Nos. 7-11, May 6-10, 1996, pp. 1123-1137. [Retrieved from http://meyerweb.com/eric/talks/www5/borealis/html Mar. 18, 2014].
Marc H. Brown and Marc A. Najork, "Distributed Active Objects," In Proceedings of the Fifth International World Wide Web Conference; Computer Networks and ISDN Systems, V.28, Nos. 7-11, May 6-10, 1996, pp. 1037-1052. [Retrieved from ftp://gatekeeper.research.compaq.com/pub/ddec/SRC/research-reports/SRC-141a.html Mar. 18, 2014].
Michael P. Plezbert and Ron K. Cytron, "Does "Just in Time" = "Better Late than Never"?" In Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Jan. 15-17, 1997, pp. 120-131.
Anawat Chankhunthod, et al., "A Hierarchical Internet Object Cache," In Proceedings of the Annual Technical Conference on USENIX 1996, Jan. 22-26, 1996, 11 pages.
E.P. Markatos, E.P., "Main Memory Caching of Web Documents," In Proceedings of the Fifth International World Wide Web Conference; Computer Networks and ISDN Systems, V. 28, Nos. 7-11, May 6-10, 1996, 15 pages. [Retrieved from http://archvlsi.ics.forth.gr/papers/www5/Overview.html Mar. 18, 2014].
M. Frans Kaashoek, et al., Server Operating Systems, In Proceedings of the 7th Workshop on ACM SIGOPS European Workshop: Systems Support for Worldwide Applications, Sep. 9-11, 1996, 8 pages.
Barron C. Housel, et al., "WebExpress: A System for Optimizing Web Browsing in a Wireless Environment," In Proceedings of the 2nd Annual International Conference on Mobile Computing and Networking, Nov. 1996, 9 pages.
Thomas T. Kwan, et al., "NCSA's World Wide Web Server: Design and Performance," Computer, V. 28 No. 11, Nov. 1995, pp. 68-74.
Jonathan Trevor, et al., "Exorcising Daemons: A Modular and Lightweight Approach to Deploying Applications on the Web," In Proceedings of the Fifth International World Wide Web Conference; Computer Networks and ISDN Systems, V. 28, Nos. 7-11, May 6-10, 1996, pp. 1053-1062. [Retrieved from http://www.ra.ethz.ch/CDstore/www5/www363/overview.htm Mar. 18, 2014].
Robert Thau, "Design Considerations for the Apache Server API," In Proceedings of the Fifth International World Wide Web Conference; Computer Networks and ISDN Systems, V. 28, Nos. 7-11, May 6-10, 1996, 13 pages. [Retrieved from http://iw3c2.cs.ust.hk/WWW5/www5conf.inria.fr/fich_html/papers/P20/Overview.html Mar. 18, 2014].
Defendant Hewlett-Packard Company's Invalidity Contentions, *Implicit Networks, Inc.*, (*Plaintiff*) v. *Hewlett-Packard Company*, (*Defendant*), United States District Court Northern District of California San Francisco Division (Jun. 30, 2011), 27 pages. [Case No. 3:10-CV-3746 SI].
Steve McCanne and Van Jacobson, "vic: A Flexible Framework for Packet Video," Proceedings of the third ACM international conference on Multimedia, ACM Multimedia 95—Electronic Proceedings, Nov. 5-9, 1995, 19 pages.
Dan Decasper et al., "Router Plugins A Software Architecture for next Generation Routers," Computer Communication Review, a publication of ACM SIGCOMM, vol. 28 No. 4, Oct. 1998, pp. 229-240.
David Mosberger, "Scout: a Path Based Operating System," Doctoral Dissertation Submitted to the University of Arizona, 1997, 174 pages.
Ion Stoica and Hui Zhang, "LIRA: An Approach for Service Differentiation in the Internet," Proceedings of the 8th International Workshop on Network and Operating Systems Support for Digital Audio and Video, Jul. 8-10, 1998, 14 pages.
Oliver Spatscheck, "Defending Against Denial of Service Attacks in Scout," Proceedings of the 3rd Symposium on Operating Systems Design and Implementation, Feb. 1999, 15 pages.
Peter Deutsch et al., "GZIP file format specification version 4.3," Network Working Group, RFC 1952, May 1996, 24 pages. [Retrieved from http://tools.ietf.org/html/rfc1952 Mar. 18, 2014].
Kamran Husain and Jason Levitt, "Javascript Developer's Resource—client-side programming using HTML, netscape plug-ins and java applets," Prentice-Hall Inc., ISBN 0-13-267923-X, 1997, pp. 16, 141, 391-393, 416-418, 420, 428.
Netscape LiveWire Developer's Guide, Version 2.0 Part No. UM151-02274-00, 1996, 190 pages.
Douglas Kramer, "The JavaTM Platform," A White Paper, JAvaSoft, May 1996, 25 pages.
Brian N. Bershad, et al., "Process for Rewriting Executable Content on a Network Server or Desktop Machine in Order to Enforce Site Specific Properties," U.S. Appl. No. 60/061,387, filed Oct. 7, 1997, pp. 1-8.
Emin Gun Sirer, et al., "Design and implementation of a distributed virtual machine for networked computers," 17th ACM Symposium on Operating System Principles (SOSP'99), Published as Operating Systems Review 34(5) Dec. 1999, pp. 202-216.
Emin Gun Sirer, Kimera: A System Architecture for Networked Computers, Department of Computer Science and Engineering, University of Washington, 1997, 2 pages.
Emin Gun Sirer, et al., Kimera Architecture, Department of Computer Science and Engineering, University of Washington 1997, 4 pages.

(56) **References Cited**

OTHER PUBLICATIONS

Emin Gun Sirer, et al., "Distributed Virtual Machines: A System Architecture for Network Computing," Department of Computer Science and Engineering, University of Washington, Feb. 26, 1998, 4 pages.

Jonathan Aldrich, et al., "Static Analyses for Eliminating Unnecessary Synchronization from Java Programs," Department of Computer Science and Engineering, University of Washington, 1999, 20 pages.

Microsoft Active Server Pages: Frequently Asked Questions, Microsoft Corporation, Sep. 1997, MS/IMPLICIT0002318, 4 pages.

Microsoft Internet Information Server—Web Server for Windows NT Operating System: Reviewer's Guide, Reviewing and Evaluating Microsoft Internet Information Server version 3.0, 1996, MS/IMPLICIT0004192, 45 pages.

Emin Gun Sirer, et al., "Improving the Security, Scalability, Manageability and Performance of System Services for Network Computing," Department of Computer Science and Engineering, University of Washington, 1998, 13 pages.

* cited by examiner

FIG. 1

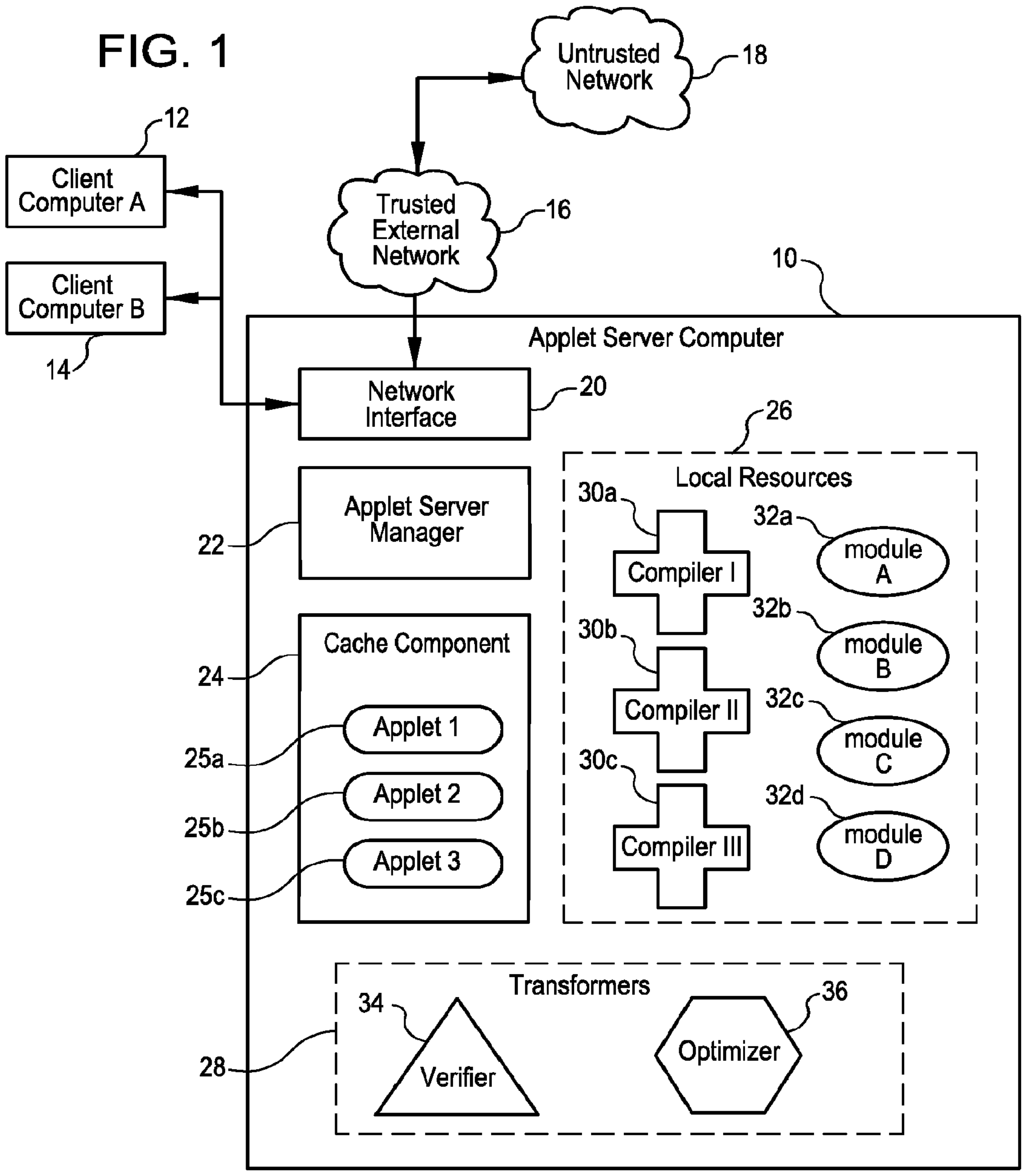

Untrusted Network
18
12
Client Computer A
Trusted External Network
16
Client Computer B
14
10
Applet Server Computer
Network Interface
20
26
Local Resources
30a
32a
Applet Server Manager
22
module A
Compiler I
32b
30b
Cache Component
24
module B
Compiler II
32c
Applet 1
25a
module C
30c
Applet 2
25b
32d
Compiler III
module D
Applet 3
25c
Transformers
34
36
28
Optimizer
Verifier

FIG. 2A

Request Data Type

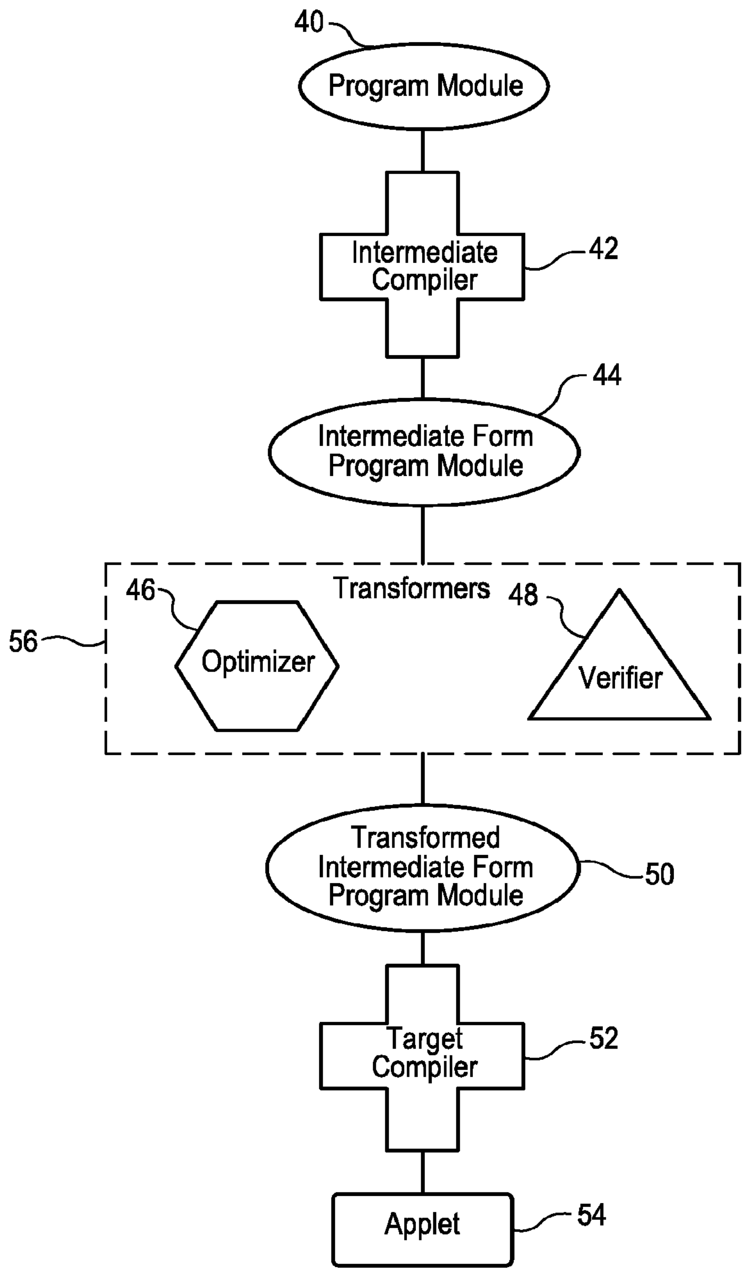
Tag	Value
Appplet - URL	(String) specifies the name of the requested applet.
Code -Type	(Source/Intermediate/Binary) specifies the format the applet is to be delivered to the requesting client in. A request for binary would specify the CPU of the requesting client (e.g., x 86)
Verification - Level	(0 - 100) specifies the degree of the verification to be performed. 0 = no/minimal verification, 100 = maximum verification (highest level of security).
Optimization - Level	(0 - 100) specifies the degree of optimization to be performed. 0 = no/minimal optimization, 100 = maximum optimization.

FIG. 2B

Code Data Type

Tag	Value
Appplet - URL	(String) specifies the name of the requested applet.
Code -Type	(Source/Intermediate/Binary) specifies the format the applet is to be delivered to the requesting client in. A request for binary would specify the CPU of the requesting client (e.g., x 86)
Verification - Level	(0 - 100) specifies the degree of the verification to be performed. 0 = no/minimal verification, 100 = maximum verification (highest level of security).
Optimization - Level	(0 - 100) specifies the degree of optimization to be performed. 0 = no/minimal optimization, 100 = maximum optimization.
Applet Length	(0 - 2^{32}) specifies the size of the requested applet.
Applet Code	The Requested Applet in the form specified by the requested data type.

FIG. 3


40
Program Module
Intermediate Compiler
42
44
Intermediate Form Program Module
Transformers
46
Optimizer
48
Verifier
56
Transformed Intermediate Form Program Module
50
Target Compiler
52
Applet
54

APPLICATION SERVER FOR DELIVERING APPLETS TO CLIENT COMPUTING DEVICES IN A DISTRIBUTED ENVIRONMENT

This application is a continuation of U.S. application Ser. No. 13/269,905, filed Oct. 10, 2011 (now U.S. Pat. No. 8,856,779), which is a continuation of U.S. application Ser. No. 11/933,161, filed Oct. 31, 2007 (now U.S. Pat. No. 8,056,075), which is a continuation of U.S. application Ser. No. 11/241,985, filed Oct. 4, 2005 (now U.S. Pat. No. 7,774,740), which is a continuation of Ser. No. 09/968,704 filed Oct. 1, 2001 (now U.S. Pat. No. 6,976,248), which is a continuation of U.S. application Ser. No. 09/040,972 filed Mar. 18, 1998 (now U.S. Pat. No. 6,324,685); the disclosures of all of the above-referenced applications are incorporated by reference herein in their entireties.

The following application is incorporated by reference in its entirety as if fully set forth herein: U.S. application Ser. No. 11/241,985 filed Oct. 4, 2005.

BACKGROUND

1. Technical Field

The present invention elates to computer operating system arid, in particular to a server architecture providing application caching and security verification.

2. Description of the Related Art

The growth of the Internet's importance to business, along with the increased dependence upon corporate networks, has created a demand for more secure and efficient computer systems. The traditional solution to this problem has been to depend upon improvements in hardware performance to make up for the performance penalty that is typically incurred when a computer system is made more secure and stable. Increased interconnectivity has also created a need for improved interoperability amongst a variety of computers that are now connected to one another. One solution to the problem of the variety of computers interconnected via the Internet and corporate networks has been the development of portable architecture neutral programming languages. The most widely known of these is the Java™ programming language, though, there are numerous other-architecture neutral languages.

Architecture neutral programming languages allow programs downloaded from a server computer to a client computer to be interpreted and executed locally. This is possible because the compiler generates partially compiled intermediate byte-code, rather than fully compiled native machine code. In order to run a program, the client machine uses an interpreter to execute the compiled byte-code. The byte-codes provide an architecture neutral object file format, which allows the code to be transported to multiple platforms. This allows the program to be run on any system which implements the appropriate interpreter and run-time system. Collectively, the interpreter and runtime system implement a virtual machine, this structure results in a very secure language.

The security of this system is premised on the ability of the byte-code to be verified independently by the client computer. Using the Java™ programming language or some other virtual machine implementing technology, a client can ensure that the downloaded program will not crash the user's computer or perform operations for which it does not have permission.

The traditional implementations of architecture neutral languages are not without problems. While providing tremendous cross platform support, the current implementations of architecture neutral languages require that every client performs its own verification and interpretation of the intermediate code. The high computation and memory requirements of a verifier, compiler and interpreter restrict the applicability of these technologies to powerful client computers.

Another problem with performing the verification process on the client computer is that any individual within an organization may disable some or of the checks performed on downloaded code. The current structure of these systems makes security management at the enterprise level almost impossible. Since upgrades of security checking software must be made on every client computer, the cost and time involved in doing such upgrades makes it likely that outdated or corrupt copies of the verifier or interpreter exist within an organization. Even when an organization is diligent in maintaining a client based security model, the size of the undertaking in a large organization increases the likelihood that there will be problems.

There is a need for a scalable distributed system architecture that provides a mechanism for client computers to request and execute applets in a safe manner without requiring the client machines to have local resources to compile or verify the code. There is a further need for a system in which the applets may be cached in either an intermediate architecture neutral form or machine specific form in order to increase overall system performance and efficiency.

SUMMARY

In accordance with one embodiment of the invention, an applet server architecture is taught which allows client computers to request and execute applets in a safe manner without requiring the client to have local resources to verify or compile the applet code. Compilation and byte code verification in the present invention are server based and thereby provide more efficient use of resources and a flexible mechanism for instituting enterprise-wide security policies. The server architecture also provides a cache for applets, allowing clients to receive applet code without having to access nodes outside the local network. The cache also provides a mechanism for avoiding repeated verification and compilation of previously requested applet code since any client requesting a given applet will have the request satisfied by a single cache entry.

Machine specific binary code is essentially interpreted code since the processor for a given computer can essentially be viewed as a form of an interpreter, interpreting binary code into the associated electronic equivalents. The present invention adds a level of indirection in the form of an intermediate language that is processor independent. The intermediate language serves as the basis for security verification, code optimizations, or any other compile time modifications that might be necessary. The intermediate form allows a single version of the source to be stored for many target platforms instead of having a different binary fir each potential target computer. Compilations to the target form can either be done at the lime of a cache hit or they can be avoided all together if the target machine is able to directly interpret the intermediate form. if the compilation is done on the server, then a copy of the of the compiled code as well as the intermediate form can be stored in the cache. The performance advantage derived from caching the compiled form as well as the intermediate depends upon the number of clients with the same CPU.

The novel features believed characteristic of the invention are set forth in the appended claims. The invention itself, however, as well as other features and advantages thereof will best be understood by reference to the detailed description which follows, when read in conjunction with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is diagram showing the major components which may be used to implement an applet server in one embodiment of the present invention;

FIG. **2***a* is a table which illustrates the structure of the request format data type;

FIG. **2***b* is a table which illustrates the structure of the returned code data type.

FIG. **3** is a diagram showing the compilation and transformation of a program module into an applet in a particular form.

DETAILED DESCRIPTION

Referring to FIG. **1**, an applet server architecture according to one embodiment of the invention is based on an applet server computer **10** which in turn is connected to client computer A **12**, client computer B **14**, an external network **16** and an untrusted network **18**. The applet server computer **10** connects to client computers **12** and **14**, an external network **16**, and an untrusted network **18** by means of a network interface **20**. Typically this connection will involve one or more of the computers or networks having a connection to the Internet.

The applet server computer **10** accomplishes its objectives by manipulating computer programs in several formats. An applet (e.g. applets 1-3, **25***a*-**25***c*) is any form of program instructions, whether in binary, source or intermediate format. In the case of this architecture, the applet code can either be a self-contained program, or it can be a code fragment associated with a larger application.

Binary format refers to processor specific machine instructions suitable for running natively on a given computing platform (also referred to as "target" because of the concept of "targeting" a compiler to produce binary code for a given processor type).

Source refers to non-binary applet code, generally in the form of higher level languages (i.e. the C™, C++™, Java™, Visual Basic™, ActiveX™, Fortran™, and Modula™ programming languages).

Intermediate format refers to a common intermediate byte-code that is produced by compiling a given source code put. The intermediate byte-code need not necessarily be Java™ byte-code.

Treating applets in this general sense allows client computers **12** and **14** to request not only applications, but portions of applications. Client computers **12** and **14** are thus able to use applet server computer **10** as the equivalent of a loader, loading in appropriate parts of the application in the form of applets. In turn client computers **12** and **14** can run large applications without requiring that the client computers **12** and **14** have the resources to store the entire application in memory at once.

Having the applets delivered from applet server computer **10** allows code in intermediate form to be verified, optimized, and compiled before being transmitted to client computers **12** and **14**. This reduces the amount of work the client computers **12** and **14** have to do and provides a convenient way to impose global restrictions on code.

In operation, client computer A **12** transmits a request to an applet server computer **10** requesting an applet in a particular form. The form may be selected from a large matrix of many possible for that can be recognized by the system. The request

specifies the format (source, intermediate, or binary) which the client wishes to receive the applet. The request may also specify that the applet be verified or have some other transformation operation preformed upon it. Verification, optimization and compression are examples of types of transformation operations. The request is received by the network interface **20** of the applet server computer **10** which passes the request onto the applet server manager **22**.

After interpreting the request, the applet server manager **22** checks to see if the requested applet is available in the cache **24**. The cache **24** stores applets in a variety of formats (source, intermediate, or binary). If the requested form of the applet is available in the cache **24** (applet 1 **25***a*, applet 2 **25***b*, or applet 3 **25***c* in this example) the applet server manager **22** instructs the network interface **20** to transmit the applet to requesting client computer A **12**.

If the requested applet is not available in the cache **24**, then the apple server manager **22** will attempt to build the requested applet from local resources **26** and one or more transformation operations performed by one or more of the transformers **28**. Local resources **26** are comprised of compilers **30***a*, **30***b* and **30***c* and program code modules **32***a*, **32***b*, **32***c* and **32***d*. The requested applet is built by selecting one or more program code modules **32** and compiling them with one or more compilers **30**. Transformer operations may be performed by the verifier **34** or the optimizer **36**. After the applet server manager **22** builds the applet, the network interface **20** transmits the applet to the requesting client computer A **12**.

If the request can not be satisfied by building the applet from local resources **26** and transformers **28**, the applet server manager **22** will pass a request for the requested applet to external network **16** and/or untrusted network **18**. The applet server manager **22** may request the applet in intermediate form or in executable form or it may request the local resources **26** and transformers **28** it needs to complete building the applet itself.

The cache **24** is capable of responding to the following commands: GET, PUT, and FLUSH. GET is used to retrieve a given applet from the cache. PUT is used to store an applet in the cache. FLUSH is used to clear the cache of one or more entries. When the cache is unable to locate an item in response to a GET operation, it returns a cache miss. The program which issued the GET command is then responsible for locating the desired form of the applet by other means and optionally storing it in the cache when it is retrieved (using the PUT operation). The FLUSH command will clear the cache of one or more entries and any subsequent GETs for the FLUSHed applet code will result in a cache miss. This is useful if a particular applet needs to be updated from a remote server on a periodic basis. When using PUT, the program issuing the command specifies a time to live (TTL) in the cache. When the TTL expires, the cache entry is removed by means of a FLUSH operation.

Local resources **26** are comprised of program modules **32** (applets in source form, not the requested form) and compilers **30**. The program modules **32** are run through the compilers **30** in order to produce applets in the requested form. The applet server manager **20** may also direct the modules **32** to be processed by a verifier **34** or another transformer such as an optimizer **36**. Program modules **32** are program code used to build applets. Program modules **32** may be stored in local resources **26** in source, binary, or intermediate formats. When an applet is built it may require the operation of one or more compilers **30** upon one or more program modules **32**. The program modules **32** may be combined and recompiled with previously cached applets and the resulting applet may also be cached for use at a future time. Additionally, program modules **32**, compilers **30** and transformers **28** (including verifiers **34** and optimizers **36**) may be distributed across a network. The applet server manager **22** may pass requests for the components it needs to build a particular applet back to the network interface **20** which in turn passes the request onto the rest of the network and may include external network **16** and untrusted network **18**.

FIG. **3** provides further illustration of how an applet is produced from local resources and transformers. In this illustration the request is for an optimized and verified applet compiled to a machine specific form. A program module **40** is compiled into an intermediate form program module **44** by an intermediate compiler **42**. The intermediate form program module **44** is then transformed by an optimizer **46** or a verifier **48**. The resulting transformed intermediate form program module **50** is then compiled by target compiler **52** into machine specific code applet **54**.

There are two types of compilers used to build applets: intermediate compilers **42** and target compilers **52**. The intermediate compiler **42** compiles program modules (source applet code) **40** and produces a common intermediate pseudo-binary representation of the source applet code (intermediate form program module **44**). The word pseudo is used because the intermediate form **44** is not processor specific but is still a binary representation of the source program module **40**. This intermediate form can be re-targeted and compiled for a particular processor. Alternatively, the intermediate form **44** can be interpreted by an interpreter or virtual machine that understands the internal binary representation of the intermediate form. A target compiler **52** compiles intermediate applet code **44** into an applet **54** in a processor specific format (binary) suitable for running natively on a given computing platform.

Transformers **56** are programs that take in intermediate code and put out intermediate code. Transformers **56** are generally used for things like verification and optimization. Other transformers might include compressors that identify portions of code that can be replaced with smaller equivalents. Transformers can be matched up to any other component that takes in intermediate code as an input. These include the cache **24** and the target compilers **52**. Global policies for transformers **56** can be implemented which ensure that all applets are run through some set of transformers before being returned to the client.

A verifier **48** is a type of transformer that is able to analyze input code and determine areas that might not be safe. The verifier **48** can determine the level of safety. Some verifiers **48** look for areas where unsafe or protected memory is being accessed, others might look for accesses to system resources such as JO devices. Once a verifier **48** determines the portion of unsafe applet code several steps can be taken. The offending code portion can be encased with new code that specifically prevents this unsafe code section from being executed. The unsafe code can be modified to be safe. The unsafe code can be flagged in such a way that a user can be warned about the possible risk of executing the code fragment. The verifier's role can therefore be summarized as determining where unsafe code exists and possibly altering the offending code to render it harmless. Verifiers **48** can operate on any format of input code, whether in source, intermediate or binary form. However, since intermediate code is a common format, it is most efficient to have a single verifier that will operate on code in this format. This eliminates the need to build specific knowledge of various source languages into the verifier. Verifiers **48** are a form of a transformer. Verifiers **48** take in intermediate code and put out verified intermediate code. Verifiers **48** are responsible for identifying non-secure portions of code in the intermediate code and modifying this code to make it secure. Security problems generally include access to memory areas that are unsafe (such as system memory, or memory outside the application space of the applet).

The choice of adding in the verification step can be left up to the client computer **12**, the applet server computer **10** (see FIG. **1**), or can be based on the network that the applet originated from. Server managers can institute global policies that affect all clients by forcing all applets to be run through the verifier **48**. Alternatively, verification can be reserved for untrusted networks (**18** in FIG. **1**), or it can be left up to the client to determine whether the verification should be performed. In the preferred embodiment, verification levels determined by the applet server **10**. In this way, a uniform security policy may be implemented from a single machine (i.e., the applet server **10**).

Optimizers **46** are another type of transformer program. Optimizers **46** analyze code, making improvements to well known code fragments by substituting in optimized but equivalent code fragments. Optimizers **46** take in intermediate code **44** and put out transformed intermediate code **50**. The transformed intermediate code **50** is functionally equivalent to the source intermediate code **44** in that they share the same structure.

Referring again to FIG. **1**, policies may be instituted on the applet server **10** that force a certain set of request parameters regardless of what the client asked for. For example, the applet server manager **22** can run the applet through a verifier **34** or optimizer **36** regardless of whether the client **12** requested this or not. Since the server **10** might have to go to an untrusted network **18** to retrieve a given applet, it evil then run this applet through the required transformers **28**, particularly the verifier **34** before returning it to the client **12**. Since clients **12** and **14** have to go through the applet server computer **10**, this ensures that clients **12** and **14** do not receive applets directly from an untrusted network **18**. In addition, since the server will be dealing directly with untrusted network **18**, it can be set up to institute policies based on the network. A trusted external network **16** may be treated differently than an untrusted network **18**. This will provide the ability to run a verifier **34** only when dealing with an entrusted network **18**, but not when dealing with a trusted external network **16**. In one embodiment, intermediate code is passed through a verifier **34** and the source of the code merely determines the level of verification applied.

The client **12** is the target computer on which the user wishes to execute an applet. The client **12** requests applets from the server **10** in a specific form. Applets can be requested in various formats including source, intermediate and binary. In addition, an applet can be requested with verification and/or other compile time operations. Optionally, the client **12** can pass a verifier to the server to provide verification, If the server **10** implements its own security, then both the client and server verifiers will be run, The verifier that is passed from the client to the server is cached at the server fix subsequent verification. The client can refer to this verifier by a server-generated handle to avoid having to pass the verifier each time an applet is requested,

Client computers **12** and **14** requesting applet code in intermediate format need to have an interpreter or virtual machine capable of interpreting the binary code in the intermediate format if the applet is to be executed on the client machine.

In the preferred embodiment, requests to the applet server are in a format similar to those of an HTTP header and are comprised of tags and values. In one embodiment, an HTTP GET method is used to make the request (though use of the HTTP protocol is not necessary to implement the present invention). The request is made up of a series of tags which specify the requested applet, the platform on which it is to be run and the type of code (source/intermediate/binary), a verification level and an optimization level. New tags and values can be added to extend functionality as needed and the applet server manager **22** will discard any tag it does not recognize. When the applet server computer **10** returns the requested applet to the requesting client computer A **12**, it will transmit the request header followed by the applet code. In this instance, the header will additionally include a field which defines the length of the applet code. FIG. **2** provides a table which illustrates the request format and the returned code format.

While this invention has been described with reference to specific embodiments, this description is not meant to limit the scope of the invention. Various modifications of the disclosed embodiments, as well as other embodiments of the invention, will be apparent to persons skilled in the art upon reference to this description. It is therefore contemplated that the appended claims will cover any such modifications or embodiments as fall within the scope of the invention.

What is claimed is:

1. A non-transitory computer-readable storage medium having stored thereon instructions that are executable to cause a computer system to perform operations comprising:

receiving, at the computer system, a first HTTP request from a first client computer for a resource, wherein the resource includes source code;

producing, by the computer system, the resource for the first client computer, wherein the producing includes:

conveying, by the computer system, a request for the resource to an external network;

receiving, at the computer system, the resource from the external network; and

performing, by the computer system, a transformation operation on the resource; and

sending, by the computer system, the produced resource to the first client computer in response to the first HTTP request.

2. The non-transitory computer-readable storage medium of claim **1**, wherein the first HTTP request is an HTTP GET request, and wherein the HTTP GET request includes an identifier that specifies the resource.

3. The non-transitory computer-readable storage medium of claim **1**, wherein the first HTTP request specifies a platform of the first client computer, and wherein the produced resource includes source code that is compatible with the platform of the first client computer.

4. The non-transitory computer-readable storage medium of claim **3**, further comprising:

receiving, at the computer system, a second HTTP request from a second client computer for the resource, wherein the second HTTP request specifies a platform of the second client computer;

producing, by the computer system, the resource for the second client computer; and

sending, by the computer system to the second client computer, the produced resource for the second client computer in response to the second HTTP request wherein the produced resource for the second client computer includes source code that is compatible with the platform of the second client computer.

5. The non-transitory computer-readable storage medium of claim **4**, further comprising storing the produced resource for the first client computer in a cache.

6. The non-transitory computer-readable storage medium of claim **5**, wherein producing, by the computer system, the resource for the second client computer includes retrieving the produced resource for the first client computer from the cache.

7. The non-transitory computer-readable storage medium of claim **4**, wherein producing, by the computer system, the resource for the second client computer includes:

conveying, by the computer system, a request for the resource to the external network;

receiving, at the computer system, the resource from the external network; and

performing, by the computer system, a transformation operation on the resource.

8. The non-transitory computer-readable storage medium of claim **7**, wherein the platform of the first client computer is different from the platform of the second client computer, and wherein the source code of the produced resource for the first client computer is not identical to the source code of the produced resource for the second client computer.

9. The non-transitory computer-readable storage medium of claim **1**, wherein the first HTTP request includes a request for a specific form of the resource, wherein the specific form of the resource is based on information stored at the first client computer, and wherein the produced resource is the specific form of the resource.

10. The non-transitory computer-readable storage medium of claim **1**, wherein the resource is a web page.

11. The non-transitory computer-readable storage medium of claim **1**, wherein the resource is an application.

12. The non-transitory computer-readable storage medium of claim **1**, wherein the source code in the produced resource is in a high-level programming language.

13. The non-transitory computer-readable storage medium of claim **1**, wherein the first client computer is configured to process the produced resource, including by causing the source code to be executed on the first client computer.

14. The non-transitory computer-readable storage medium of claim **1**, wherein the transformation operation includes compressing at least a portion of the resource.

15. The non-transitory computer-readable storage medium of claim **1**, wherein the transformation operation includes optimizing at least a portion of the resource.

16. The non-transitory computer-readable storage medium of claim **1**, wherein the transformation operation includes verifying at least a portion of the resource.

17. The non-transitory computer-readable storage medium of claim **1**, wherein the transformation operation is based on the external network.

18. The non-transitory computer-readable storage medium of claim **1**, wherein producing further includes, performing, by the computer system, an additional transformation operation on the resource.

19. A computer system, comprising:

a processor; and

a non-transitory computer-readable storage medium having stored thereon instructions that are executable by the processor to cause the computer system to perform operations comprising:

receiving, at the computer system, an HTTP request from a client computer for a resource, wherein the resource includes source code;

producing, by the computer system, the resource, wherein the producing includes:

conveying, by the computer system, a request for the resource to an external network;

receiving, at the computer system, the resource from the external network; and performing, by the computer system a transformation operation on the resource; and

sending, by the computer system, the produced resource to the client computer in response to the HTTP request.

20. A method, comprising:

receiving, at a computer system, an HTTP request from a client computer fix a resource, wherein the resource includes source code;

producing, by the computer system, the resource, wherein the producing includes:

conveying, by the computer system, a request for the resource to an external network;

receiving, at the computer system, the resource from the external network; and

performing, by the computer system, a transformation operation on the resource; and

sending, by the computer system, the produced resource to the client computer in response to the HTTP request.

* * * * *

EXHIBIT 4

US008694683B2

(12) **United States Patent**
Balassanian

(10) **Patent No.: US 8,694,683 B2**
(45) **Date of Patent: *Apr. 8, 2014**

(54) METHOD AND SYSTEM FOR DATA DEMULTIPLEXING

(71) Applicant: **Implicit Networks, Inc.**, Bellevue, WA (US)

(72) Inventor: **Edward Balassanian**, Seattle, WA (US)

(73) Assignee: **Implicit Networks, Inc.**, Bellevue, WA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **13/911,324**

(22) Filed: **Jun. 6, 2013**

(65) **Prior Publication Data**

US 2013/0266025 A1 Oct. 10, 2013

Related U.S. Application Data

(63) Continuation of application No. 13/236,090, filed on Sep. 19, 2011, now abandoned, which is a continuation of application No. 10/636,314, filed on Aug. 6, 2003, now Pat. No. 8,055,786, which is a continuation of application No. 09/474,664, filed on Dec. 29, 1999, now Pat. No. 6,629,163.

(51) **Int. Cl.**
G06F 15/16 (2006.01)

(52) **U.S. Cl.**
USPC **709/246**; 709/238; 370/466

(58) **Field of Classification Search**
USPC 709/230, 228, 246, 238; 370/395.1, 370/469, 231, 466; 379/207.02, 229; 710/33
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,298,674	A	3/1994	Yun
5,414,833	A	5/1995	Hershey et al.
5,627,997	A	5/1997	Pearson et al.
5,761,651	A	6/1998	Hasebe
5,826,027	A	10/1998	Pedersen et al.
5,835,726	A	11/1998	Shwed et al.
5,848,233	A	12/1998	Radia et al.

(Continued)

FOREIGN PATENT DOCUMENTS

EP	0817031	1/1998

OTHER PUBLICATIONS

RFC: 791, Internet Protocol: DARPA Internet Program Protocol Specification, Sep. 1981, prepared for Defense Advanced Research Projects Agency Information Processing Techniques Office by Information Sciences Institute University of Southern California, 52 pages.

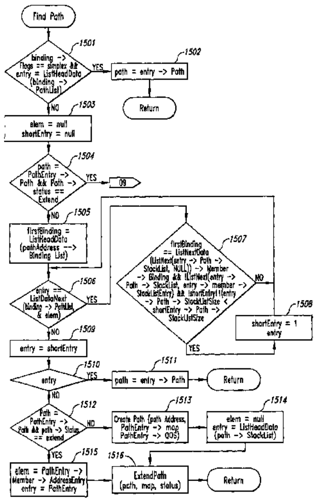
(Continued)

Primary Examiner — Jungwon Chang
(74) *Attorney, Agent, or Firm* — Meyertons, Hood, Kivlin, Kowert & Goetzel, P.C.

(57) **ABSTRACT**

A method and system for demultiplexing packets of a message is provided. The demultiplexing system receives packets of a message, identifies a sequence of message handlers for processing the message, identifies state information associated with the message for each message handler, and invokes the message handlers passing the message and the associated state information. The system identifies the message handlers based on the initial data type of the message and a target data type. The identified message handlers effect the conversion of the data to the target data type through various intermediate data types.

30 Claims, 16 Drawing Sheets


Find Path
1501
binding -> Flags == simplex && entry = ListHeadData (binding -> PathList)
YES
1502
path = entry -> Path
Return
NO
1503
elem = null shortEntry = null
1504
path = PathEntry -> Path && Path -> status == Extend
YES
09
NO
1505
firstBinding = ListHeadData (pathAddress --> Binding List)
1507
firstBinding == ListNextData (ListNext(entry -> Path -> StackList, NULL)) -> Member -> Binding && !ListNext(entry -> Path -> StackList, entry -> member -> StackListEntry) && !shortEntry!!(entry -> Path -> StackListSize < shortEntry -> Path -> StackListSize
NO
1506
entry == ListDataNext (binding -> PathList, & elem)
YES
1508
shortEntry = 1 entry
NO
1509
entry = shortEntry
YES
1510
entry
YES
1511
path = entry -> Path
Return
NO
1512
Path = PathEntry -> Path && path -> Status == extend
NO
1513
Create Path (path Address, PathEntry -> map PathEntry -> QOS)
1514
elem = null entry = ListHeadData (path -> StackList)
YES
1515
elem = PathEntry -> Member -> AddressEntry entry = PathEntry
1516
ExtendPath (path, map, status)
Return

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,848,415	A	12/1998	Guck	
5,854,899	A	12/1998	Callon et al.	
5,898,830	A	4/1999	Wesinger, Jr. et al.	
6,091,725	A	7/2000	Cheriton et al.	
6,104,500	A	8/2000	Alam et al.	
6,115,393	A *	9/2000	Engel et al.	370/469
6,119,236	A	9/2000	Shipley	
6,141,749	A	10/2000	Coss et al.	
6,151,390	A *	11/2000	Volftsun et al.	379/229
6,226,267	B1	5/2001	Spinney et al.	
6,243,667	B1	6/2001	Kerr et al.	
6,259,781	B1 *	7/2001	Crouch et al.	379/207.02
6,356,529	B1 *	3/2002	Zarom	370/231
6,401,132	B1	6/2002	Bellwood et al.	
6,426,943	B1	7/2002	Spinney et al.	
6,519,636	B2	2/2003	Engel et al.	
6,598,034	B1	7/2003	Kloth	
6,629,163	B1 *	9/2003	Balassanian	710/33
6,651,099	B1	11/2003	Dietz et al.	
6,678,518	B2	1/2004	Eerola	
6,680,922	B1	1/2004	Jorgensen	
6,701,432	B1	3/2004	Deng et al.	
6,711,166	B1 *	3/2004	Amir et al.	370/395.1
6,785,730	B1 *	8/2004	Taylor	709/230
6,871,179	B1	3/2005	Kist et al.	
6,889,181	B2	5/2005	Kerr et al.	
7,383,341	B1 *	6/2008	Saito et al.	709/228

OTHER PUBLICATIONS

Alexander, D. et al., "The SwitchWare Active Network Architecture", Jun. 6, 1998, IEEE.
Antoniazzi, S. et al., "An Open Software Architecture for Multimedia Consumer Terminals", Central Research Labs, Italy; Alcatel SEL Research Centre, Germany, ECMAST 1997.
Arbanowski, Stefan, "Generic Description of Telecommunication Services and Dynamic Resource Selection in Intelligent Communication Environments", Thesis, Technische Universitat Berlin, Oct. 9, 1996 (3 documents).
Arbanowski, S., et al., Service Personalization for Unified Messaging Systems, Jul. 6-8, 1999, The Fourth IEEE Symposium on Computers and Communications, ISCC '99, Red Sea, Egypt.
Atkinson, R., "Security Architecture for the Internet Protocol", Aug. 1995, Naval Research Laboratory.
Atkinson, R., "IP Authentication Header", Aug. 1995, Naval Research Laboratory.
Atkinson, R., "IP Encapsulating Security Payload (ESP)", Aug. 1995, Naval Research Laboratory.
Back, G., et al., Java Operating Systems: Design and Implementation, Aug. 1998, Technical Report UUCS-98-015, University of Utah.
Baker, Dr. Sean, "CORBA Implementation Issues", 1994, IONA Technologies, O'Reilly Institute Dublin, Ireland.
Barrett, R., et al., "Intermediaries: New Places for Producing and Manipulating Web Content", 1998, IBM Almaden Research Center, Elsevier Science.
Bellare, M., et al., "A Concrete Security Treatment of Symmetric Encryption: Analysis of the Des Modes of Operation", Aug. 15, 1997, Dept. of Computer Science and Engineering, University of California, San Diego.
Bellare, M., et al., "A Concrete Security Treatment of Symmetric Encryption: Analysis of the DES Modes of Operation", Aug. 15, 1997, IEEE.
Bellare, M., et al., "XOR MACs: New Methods for Message Authentication Using Finite Pseudorandom Functions", 1995, CRYPTO '95, LNCS 963, pp. 15-28, Springer-Verlag Berlin Heidelberg.
Bellissard, L., et al., "Dynamic Reconfiguration of Agent-Based Applications", Third European Research Seminar on Advances in Distributed Systems, (ERSADS '99) Madeira Island.
Bolding, Darren, "Network Security, Filters and Firewalls", 1995, www.acm.org/crossroads/xrds2-1/security.html.
Booch, G., et al., "Software Engineering with ADA", 1994, Third Edition, the Benjamin/Cummings Publishing Company, Inc. (2 documents).
Breugst, et al., "Mobile Agents—Enabling Technology for Active Intelligent Network Implementation", May/Jun. 1998, IEEE Network.
"C Library Functions", AUTH(3) Sep. 17, 1993, Solbourne Computer, Inc.
Chapman, D., et al., "Building Internet Firewalls", Sep. 1995, O'Reilly & Associates, Inc.
CheckPoint FireWall-1 Technical White Paper, Jul. 18, 1994, CheckPoint Software Technologies, Ltd.
CheckPoint FireWall-1 White Paper, Sep. 1995, Version 2.0, CheckPoint Software Technologies, Ltd.
Command Line Interface Guide P/N 093-0011-000 Rev C Version 2.5, 2000-2001, NetScreen Technologies, Inc.
Coulson, G. et al., "A CORBA Compliant Real-Time Multimedia Platform for Broadband Networks", Lecture Notes in Computer Science, 1996, Trends in Distributed Systems CORBA and Beyond.
Cox, Brad, "SuperDistribution, Objects as Property on the Electronic Frontier", 1996, Addison-Wesley Publishing Company.
Cranes, et al., "A Configurable Protocol Architecture for CORBA Environments", Autonomous Decentralized Systems 1997 Proceedings ISADS, Third International Symposium Apr. 9-11, 1997.
Curran, K., et al., "CORBA Lacks Venom", University of Ulster, Northern Ireland, UK 2000.
Dannert, Andreas, "Call Logic Service for a Personal Communication Supporting System", Thesis, Jan. 20, 1998, Technische Universitat Berlin.
DARPA Internet Program Protocol Specification, "Transmission Control Protocol", Sep. 1981, Information Sciences Institute, California.
DARPA Internet Program Protocol Specification, "Internet Protocol", Sep. 1981, Information Sciences Institute, California.
Decasper, D., et al., "Crossbow: A Toolkit for Integrated Services over Cell Switched IPv6", 1997, Computer Engineering and Networks Laboratory, ETH Zurich, Switzerland.
Decasper, D., et al., "Router Plugins A Software Architecture for Next Generation Routers", 1998, Proceedings of ACM SIGCONM '98.
Deering, S., et al., Internet Protocol, Version 6 (IPv6) Specification, Dec. 1998, Nokia, The Internet Society.
Deering, S., et al., Internet Protocol, Version 6 (IPv6) Specification, Dec. 1995, Network Working Group, RFC 1883.
Dutton, et al, "Asynchronous Transfer Mode Technical Overview (ATM)", Second Edition; IBM, Oct. 1995, 2nd Edition, Prentice Hall PTR, USA.
Eckardt, T., et al., "Application of X.500 and X.700 Standards for Supporting Personal Communications in Distributed Computing Environments", 1995, IEEE.
Eckardt, T., et al., "Personal Communications Support based on TMN and TINA Concepts", 1996, IEEE Intelligent Network Workshop (IN '96), Apr. 21-24, Melbourne, Australia.
Eckardt, T., et al., "Beyond IN and UPT—A Personal Communications Support System Based on TMN Concepts", Sep. 1997, IEEE Journal on Selected Areas in Communications, vol. 15, No. 7.
Egevang, K., et al., "The IP Network Address Translator (NAT)", May 1994, Network Working Group, RFC 1631.
Estrin, D., et al., "Visa Protocols for Controlling Inter-Organizational Datagram Flow", Dec. 1998, Computer Science Department, University of Southern California and Digital Equipment Corporation.
Faupel, M., "Java Distribution and Deployment", Oct. 9, 1997, APM Ltd., United Kingdom.
Felber, P., "The CORBA Object Group Service: A Service Approach to Object Groups in CORBA", Thesis, 1998, Ecole Polytechnique Federale de Lausanne, Switzerland.
Fish, R., et al., "DRoPS: Kernel Support for Runtime Adaptable Protocols", Aug. 25-27, 1998, IEEE 24th Euromicro Conference, Sweden.
Fiuczynski, M., et al., "An Extensible Protocol Architecture for Application-Specific Networking", 1996, Department of Computer Science and Engineering, University of Washington.

(56) **References Cited**

OTHER PUBLICATIONS

Franz, Stefan, "Job and Stream Control in Heterogeneous Hardware and Software Architectures", Apr. 1998, Technische Universitat, Berlin (2 documents).
Fraser, T., "DTE Firewalls: Phase Two Measurement and Evaluation Report", Jul. 22, 1997, Trusted Information Systems, USA.
Gazis, V., et al., "A Survey of Dynamically Adaptable Protocol Stacks", first Quarter 2010, IEEE Communications Surveys & Tutorials, vol. 12, No. 1, 1st Quarter.
Gokhale, A., et al., "Evaluating the Performance of Demultiplexing Strategies for Real-Time CORBA", Nov. 1997, GLOBECOM.
Gokhale, A., et al., "Measuring and Optimizing CORBA Latency and Scalability Over High-Speed Networks", Apr. 1998, IEEE Transaction on Computers, vol. 47, No. 4; Proceedings of the International Conference on Distributed Computing Systems (ICDCS '97) May 27-30, 1997.
Gokhale, A., et al., "Operating System Support for High-Performance, Real-Time CORBA", 1996.
Gokhale, A., et al., "Principles for Optimizing CORBA Internet Inter-ORB Protocol Performance", Jan. 9, 1998, Proceedings of the HICSS Conference, Hawaii.
Gong, Li, "Java Security: Present and Near Future", May/Jun. 1997, IEEE Micro.
Gong, Li, "New Security Architectural Directions for Java (Extended Abstract)", Dec. 19, 1996, IEEE.
Gong, Li, "Secure Java Class Loading", Nov./Dec. 1998, IEEE Internet.
Goos, G., et al., "Lecture Notes in Computer Science: Mobile Agents and Security", 1998, Springer-Verlag Berlin Heidelberg.
Goralski, W., "Introduction to ATM Networking", 1995, McGraw-Hill Series on Computer Communications, USA.
Hamzeh, K., et al., Layer Two Tunneling Protocol "L2TP", Jan. 1998, PPP Working Group, Internet Draft.
Harrison, T., et al., "The Design and Performance of a Real-Time CORBA Event Service", Aug. 8, 1997,Proceedings of the OOPSLA '97 Conference, Atlanta, Georgia in Oct. 1997.
Huitema, Christian, "IPv6 The New Internet Protocol", 1997 Prentice Hall, Second Edition.
Hutchins, J., et al., "Enhanced Internet Firewall Design Using Stateful Filters Final Report", Aug. 1997, Sandia Report; Sandia National Laboratories.
IBM, Local Area Network Concepts and Products: Routers and Gateways, May 1996.
Juniper Networks Press Release, Juniper Networks Announces Junos, First Routing Operating System for High-Growth Internet Backbone Networks, Jul. 1, 1998, Juniper Networks.
Juniper Networks Press Release, Juniper Networks Ships the Industry's First Internet Backbone Router Delivering Unrivaled Scalability, Control and Performance, Sep. 16, 1998, Juniper Networks.
Karn, P., et al., "The ESP DES-CBC Transform", Aug. 1995, Network Working Group, RFC 1829.
Kelsey, J. et al., "Authenticating Outputs of Computer Software Using a Cryptographic Coprocessor", Sep. 1996, CARDIS.
Krieger, D., et al., "The Emergence of Distributed Component Platforms", Mar. 1998, IEEE.
Krupczak, B., et al., "Implementing Communication Protocols in Java", Oct. 1998, IEEE Communications Magazine.
Krupczak, B., et al., "Implementing Protocols in Java: The Price of Portability", 1998, IEEE.
Lawson, Stephen, "Cisco NetFlow Switching Speeds Traffic Routing", Jul. 7, 1997, Infoworld.
Li, S., et al., "Active Gateway: A Facility for Video Conferencing Traffic Control", Feb. 1, 1997, Purdue University; Purdue e-Pubs; Computer Science Technical Reports.
Magedanz, T., et al., "Intelligent Agents: An Emerging Technology for Next Generation Telecommunications?", 1996, IEEE.
Mills, H., et al., "Principles of Information Systems Analysis and Design", 1986, Academic Press, Inc. (2 documents).
Mosberger, David, "*Scout: A Path-Based Operating System*", Doctoral Dissertation Submitted to the University of Arizona, 1997 (3 documents).
Muhugusa, M., et al., "ComScript: An Environment for the Implementation of Protocol Stacks and their Dynamic Reconfiguration", Dec. 1994.
Nelson, M., et al., The Data Compression Book, 2nd Edition, 1996, M&T Books, A division of MIS Press, Inc.
NetRanger User's Guide, 1996, WheelGroup Corporation.
NetScreen Command Line Reference Guide, 2000, P/N 093-0000-001 Rev A, NetScreen Technologies, Inc., USA.
NetScreen Command Line Reference Guide, 2000, P/N 093-0000-001 NetScreen Technologies, Inc., USA.
NetScreen Concepts and Examples ScreenOS Reference Guide, 1998-2001, Version 2.5 P/N 093-0039-000 Rev. A, NetScreen Technologies, Inc.
NetScreen Products Webpage, wysiwyg://body_bottom.3/http://www...een.com/products/products.html 1998-1999, NetScreen Technologies, Inc.
NetScreen WebUI, Reference Guide, Version 2.5.0 P/N 093-0040-000 Rev. A, 2000-2001, NetScreen Technologies, Inc.
NetStalker Installation and User's Guide, 1996, Version 1.0.2, Haystack Labs, Inc.
Niculescu, Dragos, "Survey of Active Network Research", Jul. 14, 1999, Rutgers University.
Nortel Northern Telecom, "ISDN Primary Rate User-Network Interface Specification", Aug. 1998.
Nygren, Erik, "The Design and Implementation of a High-Performance Active Network Node", Thesis, Feb. 1998, MIT.
Osbourne, E., "Morningstar Technologies SecureConnect Dynamic Firewall Filter User's Guide", Jun. 14, 1995, V. 1.4, Morning Star Technologies, Inc.
Padovano, Michael, "Networking Applications on UNIX System V Release 4," 1993 Prentice Hall, USA (2 documents).
Pfeifer, T., "Automatic Conversion of Communication Media", 2000, GMD Research Series, Germany.
Pfeifer, T., "Automatic Conversion of Communication Media", Thesis, 1999, Technischen Universitat Berlin, Berlin.
Pfeifer, T., et al., "Applying Quality-of-Service Parametrization for Medium-to-Medium Conversion", Aug. 25-28, 1996, 8th IEEE Workshop on Local and Metropolitan Area Networks, Potsdam, Germany.
Pfeifer, T., "Micronet Machines—New Architectural Approaches for Multimedia End-Systems", 1993 Technical University of Berlin.
Pfeifer, T., "On the Convergence of Distributed Computing and Telecommunications in the Field of Personal Communications", 1995, KiVS, Berlin.
Pfeifer, T., "Speech Synthesis in the Intelligent Personal Communication Support System (IPCSS)", Nov. 2-3, 1995, 2nd 'Speak!' Workshop on Speech Generation in Multimodal Information Systems and Practical Applications.
Pfeifer, T., et al., "Generic Conversion of Communication Media for Supporting Personal Mobility", Nov. 25-27, 1996, Proc. of the Third COST 237 Workshop: Multimedia Telecommunications and Applications.
Pfeifer, T., et al., "Intelligent Handling of Communication Media", Oct. 29-31, 1997, 6th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS) Tunis.
Pfeifer, T., et al., "Resource Selection in Heterogeneous Communication Environments using the Teleservice Descriptor", Dec. 15-19, 1997, Proceedings from the 4th COST 237 Workshop: From Multimedia Services to Network Services, Lisboa.
Pfeifer, T., et al., Mobile Guide—Location-Aware Applications from the Lab to the Market, 1998, IDMS '98, LNCS 1483, pp. 15-28.
Pfeifer, T., et al., "The Active Store providing Quality Enhanced Unified Messaging", Oct. 20-22, 1998, 5th Conference on computer Communications, AFRICOM-CCDC '98, Tunis.
Pfeifer, T.,, et al., "*A Modular Location-Aware Service and Application Platform*", 1999, Technical University of Berlin.
Plagemann, T., et al., "*Evaluating Crucial Performance Issues of Protocol Configuration in DaCaPo*", 1994, University of Oslo.
Psounis, Konstantinos, "*Active Networks: Applications, Security, Safety, and Architectures*", First Quarter 1999, IEEE Communications Surveys.

(56) **References Cited**

OTHER PUBLICATIONS

Rabiner, Lawrence, "*Applications of Speech Recognition in the Area of Telecommunications*", 1997, IEEE.
Raman, Suchitra, et al, "*A Model, Analysis, and Protocol Framework for Soft State-based Communications*", Department of EECS, University of California, Berkeley.
Rogaway, Phillip, "*Bucket Hashing and its Application to Fast Message Authentication*", Oct. 13, 1997, Department of Computer Science, University of California.
Schneier, B., et al., "*Remote Auditing of Software Outputs Using a Trusted CoProcessor*", 1997, Elsevier Paper Reprint 1999.
Tennenhouse, D., et al., "*From Internet to ActiveNet*", Laboratory of Computer Science, MIT, 1996.
Tudor, P., "*Tutorial MPEG-2 Video Compression*", Dec. 1995, Electronics & Communication Engineering Journal.
US Copyright Webpage of Copyright Title, "*IPv6: the New Internet Protocol*", by Christian Huitema, 1998 Prentice Hall.
Van der Meer, et al., "*An Approach for a 4th Generation Messaging System*", Mar. 21-23, 1999, The Fourth International Symposium on Autonomous Decentralized Systems ISADS '99, Tokyo.
Van der Meer, Sven, "*Dynamic Configuration Management of the Equipment in Distributed Communication Environments*", Thesis, Oct. 6, 1996, Berlin (3 documents).
Van Renesse, R. et al., "*Building Adaptive Systems Using Ensemble*", Cornell University Jul. 1997.
Venkatesan, R., et al., "*Threat-Adaptive Security Policy*", 1997, IEEE.
Wetherall, D., et al., "*The Active IP Option*", Sep. 1996, Proceedings of the 7th ACM SIGOPS European Workshop, Connemara, Ireland.
Welch, Terry, "*A Technique for High-Performance Data Compression*", 1984, Sperry Research Center, IEEE.
Zeletin, R. et al., "*Applying Location-Aware Computing for Electronic Commerce: Mobile Guide*", Oct. 20-22, 1998, 5th Conference on Computer Communications, AFRICOM-CCDC '98, Tunis.
Zell, Markus, "*Selection of Converter Chains by Means of Quality of Service Analysis*", Thesis, Feb. 12, 1998, Technische Universitat Berlin.
Feb. 4, 2008 Plaintiff's Original Complaint.
Aug. 26, 2008 Defendant NVIDIA Corporation's Answer to Complaint.
Aug. 26, 2008 Defendant Sun Microsystems, Inc.'s Answer to Complaint.
Aug. 27, 2008 Defendant Advanced Micro Devices, Inc.'s Answer to Complaint for Patent Infringement.
Aug. 27, 2008 RealNetworks, Inc.'s Answer to Implicit Networks, Inc.'s Original Complaint for Patent Infringement, Affirmative Defenses, and Counterclaims.
Aug. 27, 2008 Intel Corp.'s Answer, Defenses and Counterclaims.
Aug. 27, 2008 Defendant RMI Corporation's Answer to Plaintiff's Original Complaint.
Sep. 15, 2008 Plaintiff's Reply to NVIDIA Corporation's Counterclaims.
Sep. 15, 2008 Plaintiff's Reply to Sun Microsystems Inc.'s Counterclaims.
Sep. 16, 2008 Plaintiff's Reply to RealNetworks, Inc.'s Counterclaims.
Sep. 16, 2008 Plaintiff's Reply to Intel Corp.'s Counterclaims.
Dec. 10, 2008 Order granting Stipulated Motion for Dismissal with Prejudice re NVIDIA Corporation, Inc.
Dec. 16, 2008 Defendants AMD, RealNetworks, RMI, and Sun's Motion to Stay Pending the Patent and Trademark Office's Reexamination of the '163 Patent.
Dec. 29, 2008 Order granting Stipulated Motion for Dismissal without Prejudice of Claims re Sun Microsystems, Inc.
Jan. 5, 2009 Plaintiff's Opposition to Defendants AMD, RealNetworks, RMI, and Sun's Motion to Stay Pending Reexamination and Exhibit A.
Jan. 9, 2009 Reply of Defendants AMD, RealNetworks, RMI, and Sun's Motion to Stay Pending the Patent and Trademark Office's Reexamination of the '163 Patent.
Feb. 9, 2009 Order Granting Stay Pending the United States Patent and Trademark Office's Reexamination of U.S. Patent No. 6,629,163.
Feb. 17, 2009 Order Granting Stipulated Motion for Dismissal of Advanced Micro Devices, Inc. with Prejudice.
May 14, 2009 Order Granting Stipulated Motion for Dismissal of RMI Corporation with Prejudice.
Oct. 13, 2009 Order Granting Stipulated Motion for Dismissal of Claims Against and Counterclaims by Intel Corporation.
Oct. 30, 2009 Executed Order for Stipulated Motion for Dismissal of Claims Against and Counterclaims by RealNetworks, Inc.
Nov. 30, 2009 Plaintiff's Original Complaint, *Implicit* v *Microsoft*, Case No. 09-5628.
Jan. 22, 2010 Order Dismissing Case, *Implicit* v *Microsoft*, Case No. 09-5628.
Aug. 16, 2010 Plaintiff's Original Complaint, *Implicit* v *Cisco*, Case No. 10-3606.
Nov. 22, 2010 Defendant Cisco Systems, Inc's Answer and Counterclaims, *Implicit* v *Cisco*, Case No. 10-3606.
Dec. 13, 2010 Plaintiff, Implicit Networks, Inc.'s, Answer to Counterclaims, *Implicit* v *Cisco*, Case No. 10-3606.
Oct. 4, 2011 Order of Dismissal With Prejudice, *Implicit* v *Cisco*, Case No. 10-3606.
Aug. 24, 2010 Plaintiff's Original Complaint, *Implicit* v *Citrix*, Case No. 10-3766.
Dec. 1, 2010 Plaintiff's First Amended Complaint, *Implicit* v *Citrix*, Case No. 10-3766.
Jan. 14, 2011 Defendant Citrix Systems, Inc.'s Answer, Defenses and Counter-complaint for Declaratory Judgment, *Implicit* v *Citrix*, Case No. 10-3766.
Feb. 18, 2011 Plaintiff, Implicit Networks, Inc.'s, Answer to Defendants Counterclaims, *Implicit* v *Citrix*, Case No. 10-3766.
May 2, 2011 Order of Dismissal, *Implicit* v *Citrix*, Case No. 10-3766.
Jul. 30, 2010 Plaintiff's Original Complaint, *Implicit* v *F5*, Case No. 10-3365.
Oct. 13, 2010 Defendants' Answer and Counter-Complaint, *Implicit* v *F5*, Case No. 10-3365.
Nov. 3, 2010 Plaintiff's Answer to Counter-Complaint, *Implicit* v *F5*, Case No. 10-3365.
Dec. 10, 2010 Plaintiff's First Amended Complaint, *Implicit* v *F5*, Case No. 10-3365.
Jan. 14, 2011 Defendants' Answer to 1st Amended Complaint and Counterclaim, *Implicit* v *F5*, Case No. 10-3365.
Feb. 18, 2011 Plaintiff's Answer to F5's Amended Counter-Complaint, *Implicit* v *F5*, Case No. 10-3365.
Apr. 18, 2011 Defendants' Amended Answer to 1st Amended Complaint and Counter-Complaint, *Implicit* v *F5*, Case No. 10-3365.
May 5, 2011 Plaintiff's Answer to F5's Amended Counter-Complaint, *Implicit* v *F5*, Case No. 10-3365.
Jul. 22, 2011 F5 Networks, Inc.'s Invalidity Contentions, *Implicit* v *F5*, Case No. 10-3365.
Jul. 22, 2011 F5 Networks, Inc.'s Invalidity Contentions, Exhibit A, *Implicit* v *F5*, Case No. 10-3365 (31 documents).
Jul. 22, 2011 F5 Networks, Inc.'s Invalidity Contentions, Exhibit B, *Implicit* v *F5*, Case No. 10-3365.
Oct. 18, 2011 Joint Claim Construction & Pre-Hearing Statement (PR 4-3), *Implicit* v *F5*, Case No. 10-3365.
Oct. 18, 2011 Joint Claim Construction & Pre-Hearing Statement (PR 4-3) Exhibit A, *Implicit* v *F5*, Case No. 10-3365 (2 documents).
Nov. 28, 2011 Plaintiff's Opening Claim Construction Brief, *Implicit* v *F5*, Case No. 10-3365.
Nov. 29, 2011 Amended Joint Claim Construction & Pre-Hearing Statement, *Implicit* v *F5*, Case No. 10-3365.
Nov. 29, 2011 Amended Joint Claim Construction & Pre-Hearing, Exhibit A, *Implicit* v *F5*, Case No. 10-3365.
Dec. 12, 2011 Defendants' Claim Construction Brief, *Implicit* v *F5*, Case No. 10-3365.
Dec. 19, 2011 Plaintiff's Reply to Defendants' (F5, HP, Juniper) Responsive Claim Construction Brief (4-5), *Implicit* v *F5*, Case No. 10-3365.
Jan. 27, 2012 Transcript of Proceeding Held on Jan. 17, 2012; *Implicit* v *F5*, Case No. 10-3365.

(56) **References Cited**

OTHER PUBLICATIONS

Jan. 27, 2012 Transcript of Proceeding Held on Jan. 18, 2012; *Implicit* v *F5*, Case No. 10-3365.
Jan. 27, 2012 Transcript of Proceeding Held on Jan. 19, 2012; *Implicit* v *F5*, Case No. 10-3365.
Feb. 29, 2012 Claim Construction Order.
Aug. 15, 2012 Storer Invalidity Report.
Sep. 10, 2012 Implicit's Expert Report of Scott M. Nettles.
Mar. 13, 2013 Order Granting Defendants' Motion for Summary Judgment.
Apr. 9, 2013 Notice of Appeal to the Federal Circuit.
Aug. 23, 2010 Plaintiff's Original Complaint, *Implicit* v *HP*, Case No. 10-3746.
Nov. 23, 2010 Plaintiff's First Amended Complaint, *Implicit* v *HP*, Case No. 10-3746.
Jan. 14, 2011 Defendant HP's Answer and Counterclaims, *Implicit* v *HP*, Case No. 10-3746.
Feb. 18, 2011 Implicit Networks, Inc.'s Answer to HP Counter-claims, *Implicit* v *HP*, Case No. 10-3746.
May 10, 2011 Plaintiff's Amended Disclosure of Asserted Claims and Infringement Contentions, Case No. 10-3746.
Jun. 30, 2011 Defendant HP Company's Invalidity Contentions, *Implicit* v *HP*, Case No. 10-3746.
Jun. 30, 2011 Defendant HP Company's Invalidity Contentions, A1-14, *Implicit* v *HP*, Case No. 10-3746.
Jun. 30, 2011 Defendant HP Company's Invalidity Contentions, B1-21, *Implicit* v *HP*, Case No. 10-3746.
Sep. 20, 2010 Plaintiff's Original Complaint, *Implicit* v *Juniper*, Case No. 10-4234.
Nov. 12, 2010 Juniper Network's Motion to Dismiss for Failure to State a Claim Under Rule 12(B)(6): Memorandum of Points and Authorities; *Implicit* v *Juniper*, Case No. 10-4234.
Nov. 2, 2010 Juniper Network's Request for Judicial Notice in Support of its Motion to Dismiss for Failure to State a Claim Under Rule 12(B)(6): Memorandum of Points and Authorities; *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 1, 2010 First Amended Complaint; *Implicit* v *Juniper*, Case No. 10-4234.
Jan. 18, 2011 Juniper Networks, Inc.'s Answer and Affirmative Defenses to 1st Amended Complaint, *Implicit* v *Juniper*, Case No. 10-4234.
Feb. 18, 2011 Plaintiff's Answer to Defendant's Counterclaims, *Implicit* v *Juniper*, Case No. 10-4234.
May 23, 2011 Plaintiff's Disclosure of Asserted Claims and Infringement Contentions, *Implicit* v *Juniper*, Case No. 10-4234.
Nov. 15, 2011 Plaintiff's Amended Disclosure of Asserted Claim and Infringement Contentions, *Implicit* v *Juniper*, Case No. 10-4234.
Nov. 28, 2011 Spencer Hosie Declaration in Support of Plaintiff's Opening Claim Construction Brief), *Implicit* v *Juniper*, Case No. 10-4234.
Nov. 28, 2011 Spencer Hosie Declaration in Support of Plaintiff's Opening Claim Construction Brief Exhibit E, *Implicit* v *Juniper*, Case No. 10-4234.
Nov. 28, 2011 Spencer Hosie Declaration in Support of Plaintiff's Opening Claim Construction Brief Exhibit J, *Implicit* v *Juniper*, Case No. 10-4234.
Nov. 28, 2011 Spencer Hosie Declaration in Support of Plaintiff's Opening Claim Construction Brief Exhibit K, *Implicit* v *Juniper*, Case No. 10-4234.
Nov. 28, 2011 Spencer Hosie Declaration in Support of Plaintiff's Opening Claim Construction Brief Exhibits M-O, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit B, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit F, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit N, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit P, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit Q, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit S., *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit T-1, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit T-2, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit T-3, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit T-4, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit U, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit V, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit W, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit X, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit Y-1, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit Y-2, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit Y-3, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit Y-4, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit Z, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 19, 2011 Spencer Hosie Declaration in Support of Plaintiff's Reply Claim Construction Brief, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 19, 2011 Spencer Hosie Declaration in Support of Plaintiff's Reply Claim Construction Brief, Exhibit P, *Implicit* v *Juniper*, Case No. 10-4234.
Jan. 10, 2012 Plaintiff's Jan. 10, 2012 Amended Disclosure of Asserted Claims and Infringement Contentions, *Implicit* v *Juniper*, Case No. 10-4234.
Feb. 10, 2012 Juniper Networks, Inc.'s Supplemental Invalidity Contentions, *Implicit* v *Juniper*, Case No. 10-4234.
Feb. 10, 2012 Juniper Networks, Inc.'s Supplemental Invalidity Contentions, Exhibit A1, *Implicit* v *Juniper*, Case No. 10-234.
Feb. 10, 2012 Juniper Networks, Inc.'s Supplemental Invalidity Contentions, Exhibit A2, *Implicit* v *Juniper*, Case No. 10-4234.
Feb. 10, 2012 Juniper Networks, Inc.'s Supplemental Invalidity Contentions, Exhibit A3, *Implicit* v *Juniper*, Case No. 10-4234.
Feb. 10, 2012 Juniper Networks, Inc.'s Supplemental Invalidity Contentions, Exhibit A4, *Implicit* v *Juniper*, Case No. 10-4234.
Feb. 10, 2012 Juniper Networks, Inc.'s Supplemental Invalidity Contentions, Exhibit B1, *Implicit* v *Juniper*, Case No. 10-4234.

(56) **References Cited**

OTHER PUBLICATIONS

Feb. 29, 2012 Plaintiff's Feb. 29, 2012 Amended Disclosure of Asserted Claims and Infringement Contentions, *Implicit* v *Juniper*, Case No. 10-4234.
Apr. 6, 2012 Plaintiff's Apr. 6, 2012 Amended Disclosure of Asserted Claims and Infringement Contentions, *Implicit* v *Juniper*, Case No. 10-4234.
Apr. 9, 2012 Plaintiff's Apr. 9, 2012 Amended Disclosure of Asserted Claims and Infringement Contentions, *Implicit* v *Juniper*, Case No. 10-4234.
Sep. 11, 2012 Implicit's Expert Report of Scott Nettles.
Nov. 9, 2012 Juniper's Notice of Motion and Memorandum of Law ISO Motion for Summary Judgment or, in the alternative, for Partial Summary Judgment, on the Issue of Invalidity.
Nov. 9, 2012 Exhibit 2 to Declaration in support of Juniper's Motion for Summary Judgment—Calvert Expert Report.
Nov. 9, 2012 Exhibit 3 to Declaration in support of Juniper's Motion for Summary Judgment—Calvert Supplemental Expert Report.
Nov. 26, 2012 Implicit Opposition to Juniper's and F5 Motion on Invalidity.
Nov. 26, 2012 Exhibit A to Hosie Declaration—Aug. 27, 2012 Excerpts from David Blaine deposition.
Nov. 26, 2012 Exhibit B to Hosie Declaration—Oct. 25, 2012 Excerpts from Kenneth Calvert Deposition.
Nov. 26, 2012 Exhibit C to Hosie Declaration—Aug. 15, 2012 Excerpts from Kenneth Calvert Expert Report.
Nov. 26, 2012 Exhibit D to Hosie Declaration—USPN 6,651,099 to Dietz et al.
Nov. 26, 2012 Exhibit E to Hosie Declaration—Understanding Packet-Based and Flow-Based Forwarding.
Nov. 26, 2012 Exhibit F to Hosie Declaration—Wikipedia on Soft State.
Nov. 26, 2012 Exhibit G to Hosie Declaration—Sprint Notes.
Nov. 26, 2012 Exhibit H to Hosie Declaration—Implicit's Supplemental Response to Juniper's 2nd Set of Interrogatories.
Nov. 26, 2012 Exhibit I to Hosie Declaration—USPN 7,650,634 (Zuk).
U.S. Appl. No. 11/933,022 Utility Application filed Oct. 31, 2007.
U.S. Appl. No. 11/933,022 Preliminary Amendment filed Feb. 19, 2008.
U.S. Appl. No. 11/933,022 Office Action mailed Jun. 24, 2009.
U.S. Appl. No. 11/933,022 Amendment filed Sep. 24, 2009.
U.S. Appl. No. 11/933,022 Office Action dated Dec. 11, 2009.
U.S. Appl. No. 11/933,022 Amendment and Response dated Jan. 29, 2010.
U.S. Appl. No. 11/933,022 Notice of Allowance dated Mar. 2, 2010.
U.S. Appl. No. 11/933,022 Issue Notification dated May 4, 2010.
U.S. Appl. No. 11/636,314 Utility Application filed Aug. 6, 2003.
U.S. Appl. No. 11/636,314 Office Action dated Apr. 7, 2008.
U.S. Appl. No. 11/636,314 Response to Restriction Requirement dated Aug. 5, 2008.
U.S. Appl. No. 11/636,314 Office Action dated Oct. 3, 2008.
U.S. Appl. No. 11/636,314 Response to Office Action dated Apr. 3, 2009.
U.S. Appl. No. 11/636,314 Notice of Non-Compliant Amendment dated May 4, 2009.
U.S. Appl. No. 11/636,314 Amendment to Office Action Response dated Jun. 4, 2009.
U.S. Appl. No. 11/636,314 Notice of Non-Compliant Amendment dated Jun. 12, 2009.
U.S. Appl. No. 11/636,314 Amendment to Office Action dated Jul. 10, 2009.
U.S. Appl. No. 11/636,314 Final Rejection Office Action dated Oct. 21, 2009.
U.S. Appl. No. 11/636,314 Amendment after Final Office Action dated Dec. 14, 2009.
U.S. Appl. No. 11/636,314 Advisory Action dated Jan. 11, 2010.
U.S. Appl. No. 11/636,314 Notice of Non-Compliant Amendment dated Jan. 11, 2010.
U.S. Appl. No. 11/636,314 Supplemental Amendment and Response dated Mar. 13, 2010.
U.S. Appl. No. 11/636,314 Office Action dated May 11, 2010.
U.S. Appl. No. 11/636,314 Amendment and Response dated Sep. 13, 2010.
U.S. Appl. No. 11/636,314 Final Rejection dated Nov. 24, 2010.
U.S. Appl. No. 11/636,314 Notice of Appeal dated May 19, 2011.
U.S. Appl. No. 11/636,314 Amendment and Request for Continued Examination dated Jul. 19, 2011.
U.S. Appl. No. 11/636,314 Notice of Allowance dated Sep. 13, 2011.
U.S. Appl. No. 11/636,314 Notice of Allowance dated Sep. 19, 2011.
U.S. Appl. No. 11/636,314 Issue Notification dated Oct. 19, 2011.
U.S. Appl. No. 19/474,664 Utility Application filed Dec. 29, 1999.
U.S. Appl. No. 19/474,664 Office Action dated Sep. 23, 2002.
U.S. Appl. No. 19/474,664 Amendment and Response dated Feb. 24, 2003.
U.S. Appl. No. 19/474,664 Notice of Allowance dated May 20, 2003.
U.S. Appl. No. 90/010,356 Request for Ex Parte Reexamination dated Dec. 15, 2008.
U.S. Appl. No. 90/010,356 Office Action Granting Reexamination dated Jan. 17, 2009.
U.S. Appl. No. 90/010,356 First Office Action dated Jul. 7, 2009.
U.S. Appl. No. 90/010,356 First Office Action Response dated Sep. 1, 2009.
U.S. Appl. No. 90/010,356 Patent Owner Interview Summary dated Oct. 23, 2009.
U.S. Appl. No. 90/010,356 Office Action Final dated Dec. 4, 2009.
U.S. Appl. No. 90/010,356 Amendment and Response to Office Action dated Dec. 18, 2009.
U.S. Appl. No. 90/010,356 Amendment and Response to Office Action dated Jan. 4, 2010.
U.S. Appl. No. 90/010,356 Advisory Action dated Jan. 21, 2010.
U.S. Appl. No. 90/010,356 Amendment and Response to Advisory Action dated Feb. 8, 2010.
U.S. Appl. No. 90/010,356 Notice of Intent to Issue a Reexam Certificate dated Mar. 2, 2010.
U.S. Appl. No. 90/010,356 Reexamination Certificate Issued dated Jun. 22, 2010.
U.S. Appl. No. 95/000,659 Inter Partes Reexam Request dated Feb. 13, 2012.
U.S. Appl. No. 95/000,659 Order Granting Reexamination dated Apr. 3, 2012.
U.S. Appl. No. 95/000,659 Office Action dated Apr. 3, 2012.
U.S. Appl. No. 95/000,659 Office Action Response dated Jun. 4, 2012 (including Exhibits 1 & 2) (4 documents).
U.S. Appl. No. 95/000,659 Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012.
U.S. Appl. No. 95/000,659 Appendix R-1 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Declaration of Prof. Dr. Bernhard Plattner).
U.S. Appl. No. 95/000,659 Appendix R-2 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Prof. Dr. Bernhard Plattner CV).
U.S. Appl. No. 95/000,659 Appendix R-3 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Listing of Publications to Prof. Dr. Bernhard Plattner updated Feb. 2012).
U.S. Appl. No. 95/000,659 Appendix R-4 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012(Office Action Granting Reexamination in U.S. Appl. No. 95/000,660 dated May 10, 2012).
U.S. Appl. No. 95/000,659 Appendix R-5 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Office Action in U.S. Appl. No. 95/000,660 dated May 10, 2012).
U.S. Appl. No. 95/000,659 Appendix R-6 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Implicit Networks, Inc. USPN 6,629,163 Claims Chart).
U.S. Appl. No. 95/000,659 Appendix R-7 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Internet Protocol DARPA Internet Program Protocol Specification dated Sep. 1991).

(56) **References Cited**

OTHER PUBLICATIONS

U.S. Appl. No. 95/000,659 Appendix R-8 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Atkinson, IP Encapsulating Security Payload (ESP) dated Aug. 1995).
U.S. Appl. No. 95/000,659 Appendix R-9 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Claim Construction Order dated Feb. 29, 2012).
U.S. Appl. No. 95/000,659 Appendix R-10-1 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (vol. I of Edward Balassanian Deposition Transcript dated May 30, 2012).
U.S. Appl. No. 95/000,659 Appendix R-10-2 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (vol. II of Edward Balassanian Deposition Transcript dated May 31, 2012).
U.S. Appl. No. 95/000,659 Appendix R-10-3 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (vol. III of Edward Balassanian Deposition Transcript dated Jun. 7, 2012).
U.S. Appl. No. 95/000,659 Appendix R-10-4 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (vol. IV of Edward Balassanian Deposition Transcript dated Jun. 8, 2012).
U.S. Appl. No. 95/000,659 Appendix R-11 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Implicit Networks, Inc.'s Response to Juniper Networks, Inc.'s First Set of Requests for Admission 1-32).
U.S. Appl. No. 95/000,659 Action Closing Prosecution dated Oct. 1, 2012.
U.S. Appl. No. 95/000,659 Petition to Withdraw and Reissue Action Closing Prosecution dated Nov. 20, 2012.
U.S. Appl. No. 95/000,659 Patent Owner Comments to Action Closing Prosecution dated Dec. 3, 2012.
U.S. Appl. No. 95/000,659 Opposition to Petition dated Dec. 17, 2012.
U.S. Appl. No. 95/000,659 Third Party Comments to Action Closing Prosecution dated Jan. 2, 2013.
U.S. Appl. No. 95/000,660 Inter Partes Reexam Request dated Mar. 2, 2012.
U.S. Appl. No. 95/000,660 Order Granting Reexamination dated May 10, 2012.
U.S. Appl. No. 95/000,660 Office Action dated May 10, 2012.
U.S. Appl. No. 95/000,660 Response to Office Action dated Jul. 10, 2012 (including Exhibits 1 and 2).
U.S. Appl. No. 95/000,660 Third Party Comments to Office After Patent Owner's Response dated Aug. 8, 2012 (including Revised Comments).
U.S. Appl. No. 95/000,660 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Declaration of Prof. Dr. Bernhard Plattner).
U.S. Appl. No. 95/000,660 Appendix R-1 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Prof. Dr. Bernhard Plattner CV).
U.S. Appl. No. 95/000,660 Appendix R-3 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Listing of Publications to Prof. Dr. Bernhard Plattner updated Feb. 2012).
U.S. Appl. No. 95/000,660 Appendix R-4 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012(Office Action Granting Reexamination in U.S. Appl. No. 95/000,660 dated May 10, 2012).
U.S. Appl. No. 95/000,660 Appendix R-5 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Office Action in U.S. Appl. No. 95/000,660 dated May 10, 2012).
U.S. Appl. No. 95/000,660 Appendix R-6 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Implicit Networks, Inc. USPN 6,629,163 Claims Chart).
U.S. Appl. No. 95/000,660 Appendix R-7 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Internet Protocol DARPA Internet Program Protocol Specification dated Sep. 1991).
U.S. Appl. No. 95/000,660 Appendix R-8 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Atkinson, IP Encapsulating Security Payload (ESP) dated Aug. 1995).
U.S. Appl. No. 95/000,660 Appendix R-9 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Claim Construction Order dated Feb. 29, 2012).
U.S. Appl. No. 95/000,660 Appendix R-10 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (vol. I-IV of Edward Balassanian Deposition Transcript dated May 30, 2012).
U.S. Appl. No. 95/000,660 Appendix R-11 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Shacham, A., et al, "*IP Payload Compression Protocol*", Network Working Group, RFC 3173 Sep. 2001).
U.S. Appl. No. 95/000,660 Appendix R-12 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Shacham, A., et al, "*IP Payload Compression Protocol*", Network Working Group, RFC 2393 Dec. 1998).
U.S. Appl. No. 95/000,660 Appendix R-13 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 ('163 Pfeiffer Claim Chart).
U.S. Appl. No. 95/000,660 Appendix R-14 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Ylonen, T., "*SSH Transport Layer Protocol*", Network Working Group—Draft Feb. 22, 1999).
U.S. Appl. No. 95/000,660 Appendix R-15 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Dommety, G., "*Key and Sequence Number Extensions to GRE*", Network Working Group, RFC 2890 Sep. 2000).
U.S. Appl. No. 95/000,660 Appendix R-16 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Monsour, R., et al, "*Compression in IP Security*" Mar. 1997).
U.S. Appl. No. 95/000,660 Appendix R-17 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Friend, R., Internet Working Group RFC 3943 dated Nov. 2004 *Transport Layer Security Protocol Compression Using Lempel-Ziv-Stac*).
U.S. Appl. No. 95/000,660 Appendix R-18 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012(Implicit Networks, Inc.'s Response to Juniper Networks, Inc.'s First Set of Requests for Admission 1-32).
U.S. Appl. No. 95/000,660 Revised—Third Party Comments to Office After Patent Owner's Response dated Nov. 2, 2012.
U.S. Appl. No. 95/000,660 Action Closing Prosecution dated Dec. 21, 2012.
U.S. Appl. No. 95/000,660 Comments to Action Closing Prosecution dated Feb. 21, 2013 (including Dec of Dr. Ng).
U.S. Appl. No. 95/000,660 Third Party Comments to Action Closing Prosecution dated Mar. 25, 2013.
PCT/US00/33634—PCT application (WO 01/2077 A2—Jul. 12, 2001).
PCT/US00/33634—Written Opinion (WO 01/50277 A3—Feb. 14, 2002).
PCT/US00/33634—International Search Report (Oct. 9, 2001).
PCT/US00/33634—Response to Official Communication dated Dec. 7, 2001 (Mar. 21, 2002).
PCT/US00/33634—International Preliminary Examination Report (Apr. 8, 2002).
PCT/US00/33634—Official Communication (Jan. 24, 2003).
PCT/US00/33634—Response to Official Communication dated Jan. 24, 2003 (Mar. 12, 2003).
PCT/US00/33634—Official Communication (May 13, 2004).
PCT/US00/33634—Response to Summons to Attend Oral Proceeding dated May 13, 2004 (Oct. 9, 2004).
PCT/US00/33634—Decision to Refuse a European Patent application (Nov. 12, 2004).
PCT/US00/33634—Minutes of the oral proceedings before the Examining Division (Oct. 12, 2004).

(56) **References Cited**

OTHER PUBLICATIONS

PCT/US00/33634—Closure of the procedure in respect to Application No. 00984234.5-2212 (Feb. 22, 2005).

May 3, 2013 Expert Report of Dr. Alfonso Cardenas Regarding Validity of U.S. Patent Nos. 6,877,006; 7,167,864; 7,720,861; and 8,082,268 (6 documents).

Expert Report of Dr. Alfonso Cardenas Regarding Validity of U.S. Patent No. 7,167,864 (3 documents).

"InfoReports User Guide: Version 3.3.1;" Platinum Technology, Publication No. PRO-X-331-UG00-00, printed Apr. 1998; pp. 1-430.

Non-Final Office Action in Inter Partes Reexamination Control No. 95/000,659 issued Aug. 16, 2013, 107 pages.

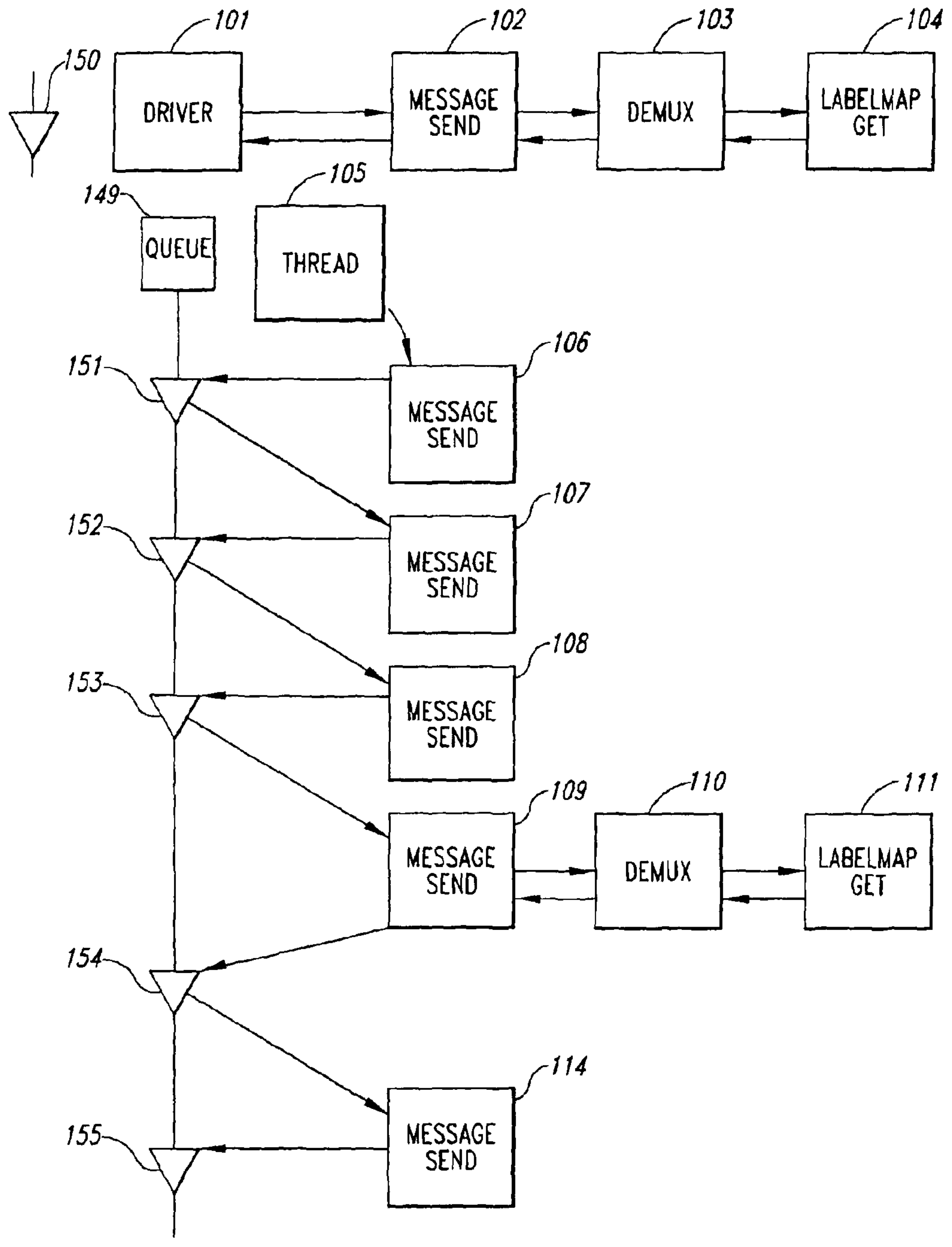
Decision on Petition in Reexamination Control No. 951000,659 issued Aug. 19, 2013, 3 pages.

Response to Non-Final Office Action in Reexamination Control No. 95/000,659 mailed Oct. 2, 2013 including Exhibits A-C, 37 pages.

Decision on Petition in Reexamination Control No. 95/000,660 issued Jul. 30, 2013, 12 pages.

Non-Final Office Action in Inter Partes Reexamination Control No. 95/000,660 issued Aug. 30, 2013, 23 pages.

* cited by examiner



150
101
DRIVER
102
MESSAGE SEND
103
DEMUX
104
LABELMAP GET
149
QUEUE
105
THREAD
106
MESSAGE SEND
151
107
MESSAGE SEND
152
108
MESSAGE SEND
153
109
MESSAGE SEND
110
DEMUX
111
LABELMAP GET
154
114
MESSAGE SEND
155

Fig. 1

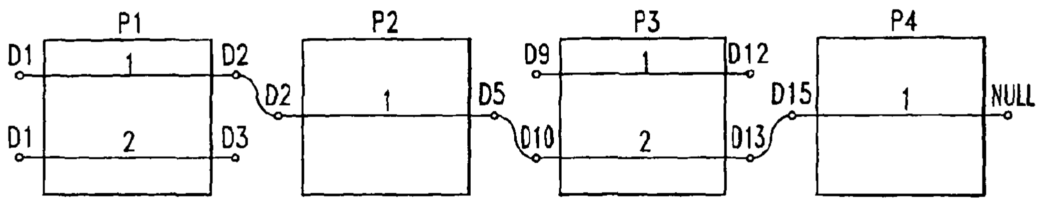

P1
P2
P3
P4
D1
1
D2
D1
2
D3
D2
1
D5
D9
1
D12
D10
2
D13
D15
1
NULL

Fig. 2

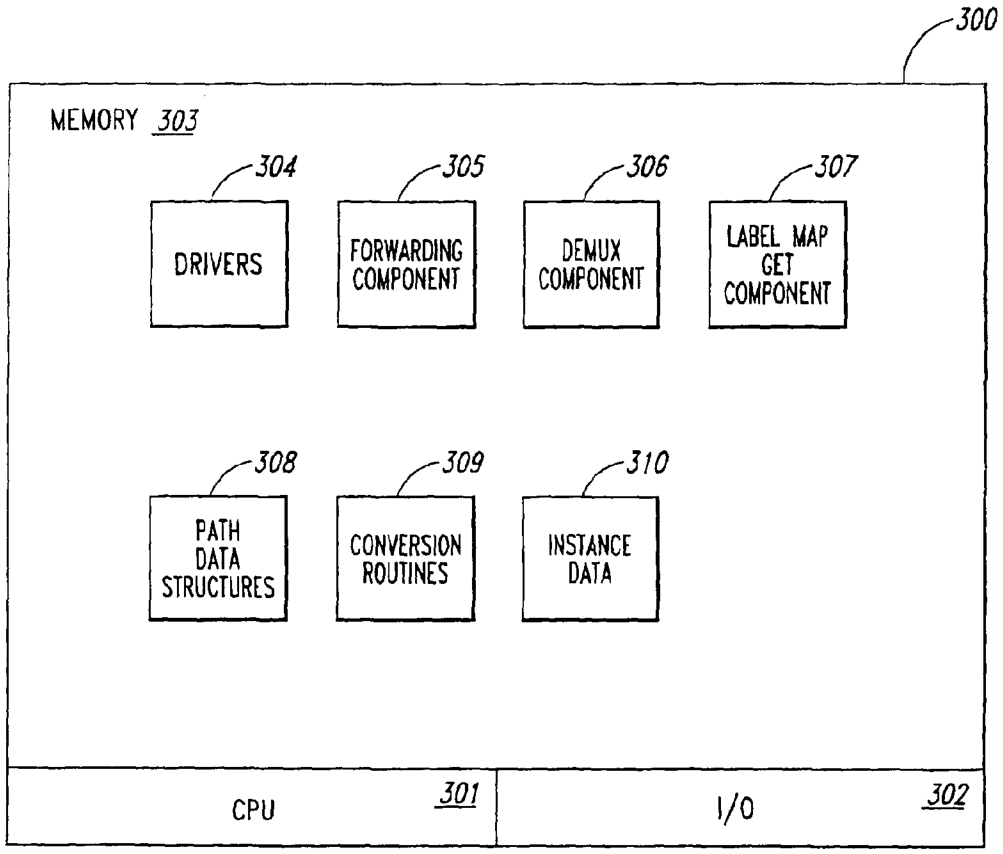

300
MEMORY 303
304
DRIVERS
305
FORWARDING COMPONENT
306
DEMUX COMPONENT
307
LABEL MAP GET COMPONENT
308
PATH DATA STRUCTURES
309
CONVERSION ROUTINES
310
INSTANCE DATA
CPU 301
I/O 302

Fig. 3

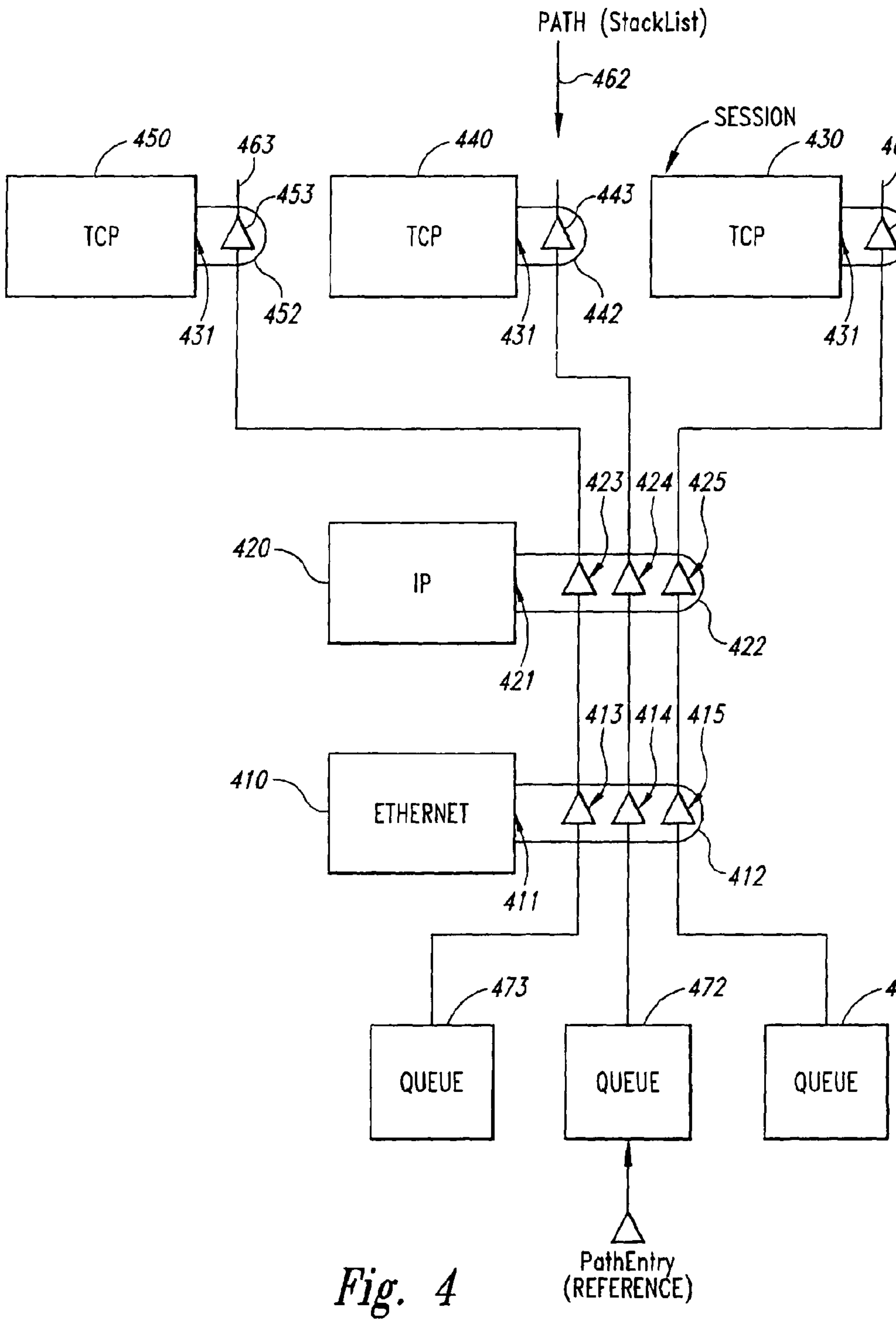

PATH (StackList)
462
SESSION
450
463
440
430
464
453
443
433
TCP
TCP
TCP
431
452
431
442
431
432
423
424
425
420
IP
422
421
413
414
415
410
ETHERNET
412
411
473
472
471
QUEUE
QUEUE
QUEUE
PathEntry
(REFERENCE)

Fig. 4

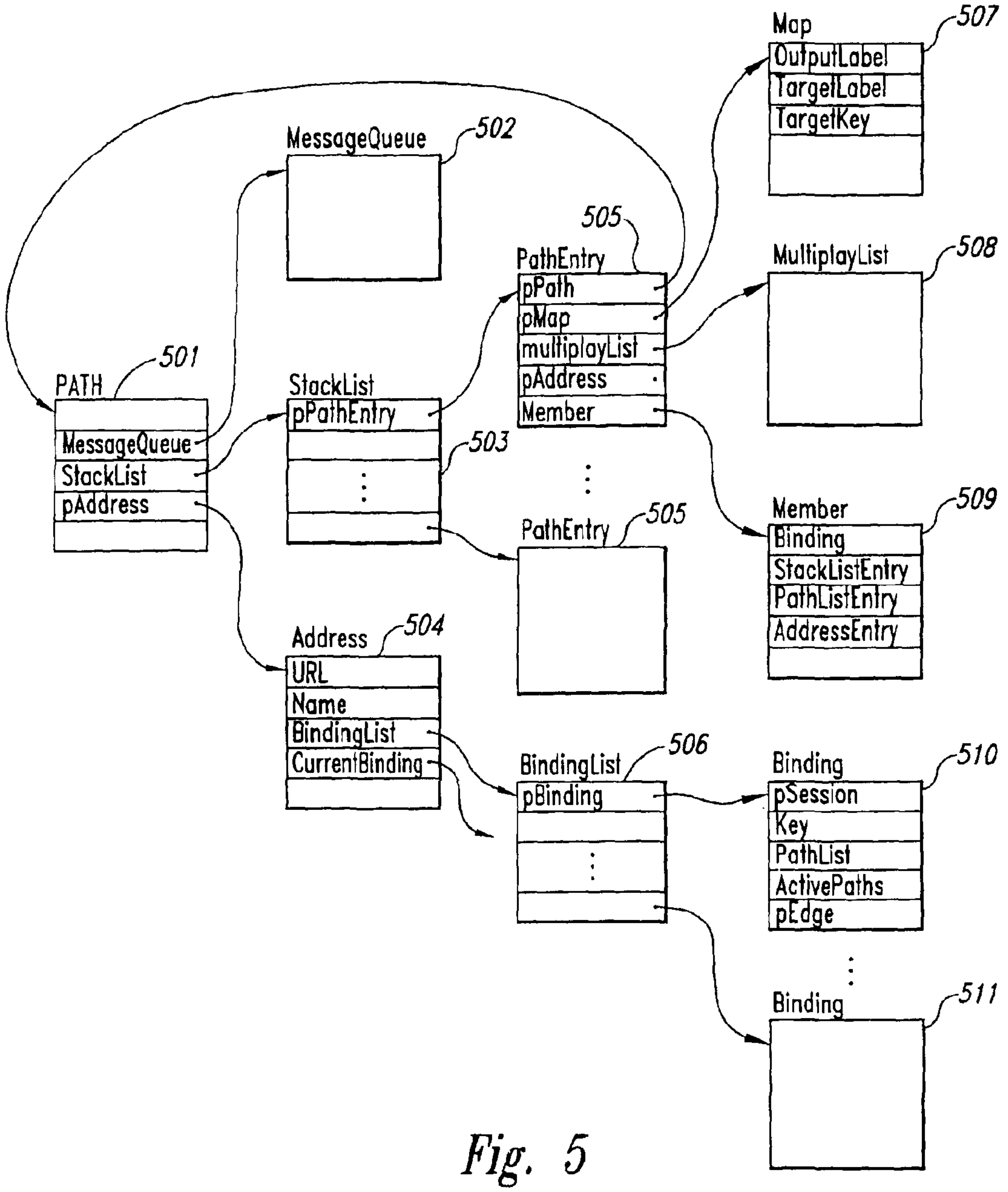

Map
507
OutputLabel
TargetLabel
TargetKey
MessageQueue
502
505
PathEntry
pPath
pMap
multiplayList
pAddress
Member
MultiplayList
508
PATH
501
MessageQueue
StackList
pAddress
StackList
pPathEntry
503
PathEntry
505
Member
509
Binding
StackListEntry
PathListEntry
AddressEntry
Address
504
URL
Name
BindingList
CurrentBinding
BindingList
506
pBinding
Binding
510
pSession
Key
PathList
ActivePaths
pEdge
Binding
511

Fig. 5

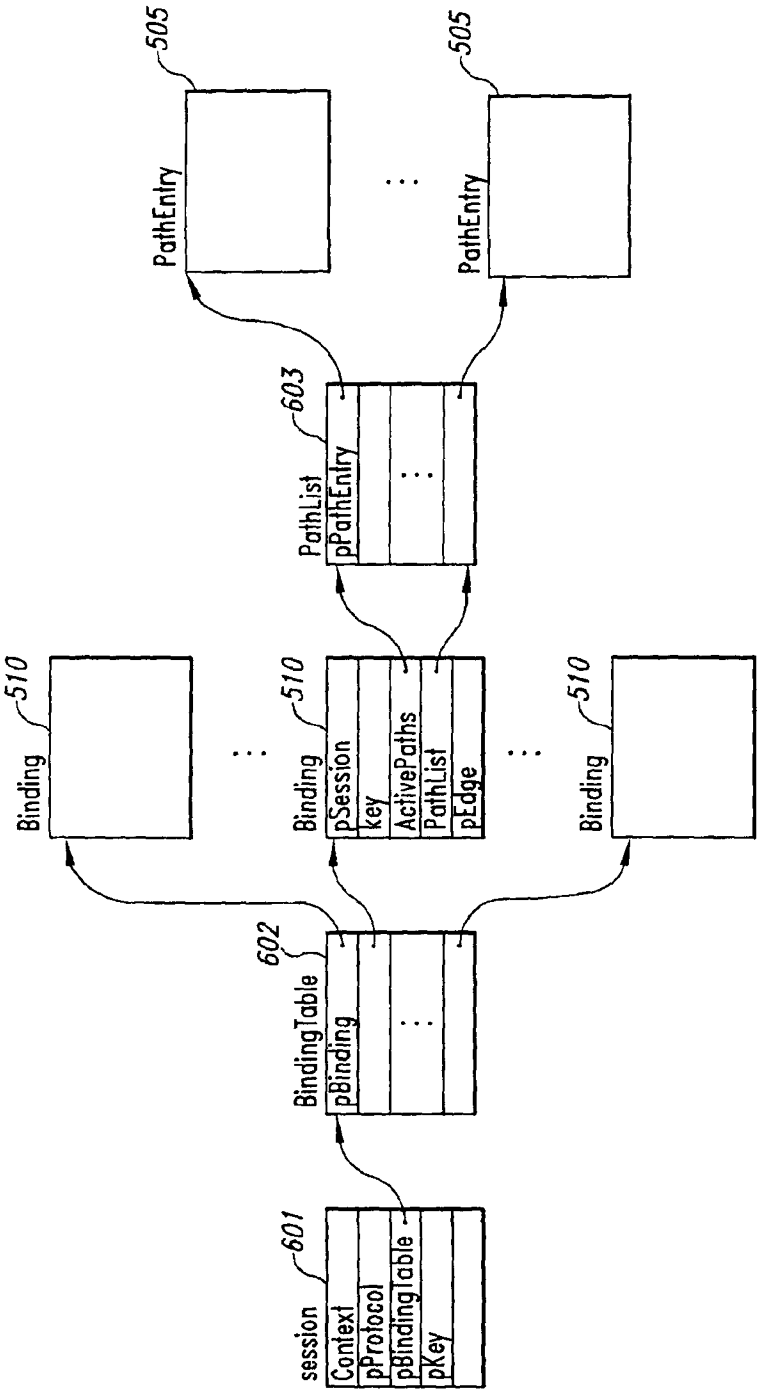

505
PathEntry
505
PathEntry
603
PathList
pPathEntry
510
Binding
510
Binding
pSession
key
ActivePaths
PathList
pEdge
510
Binding
602
BindingTable
pBinding
601
session
Context
pProtocol
pBindingTable
pKey

Fig. 6

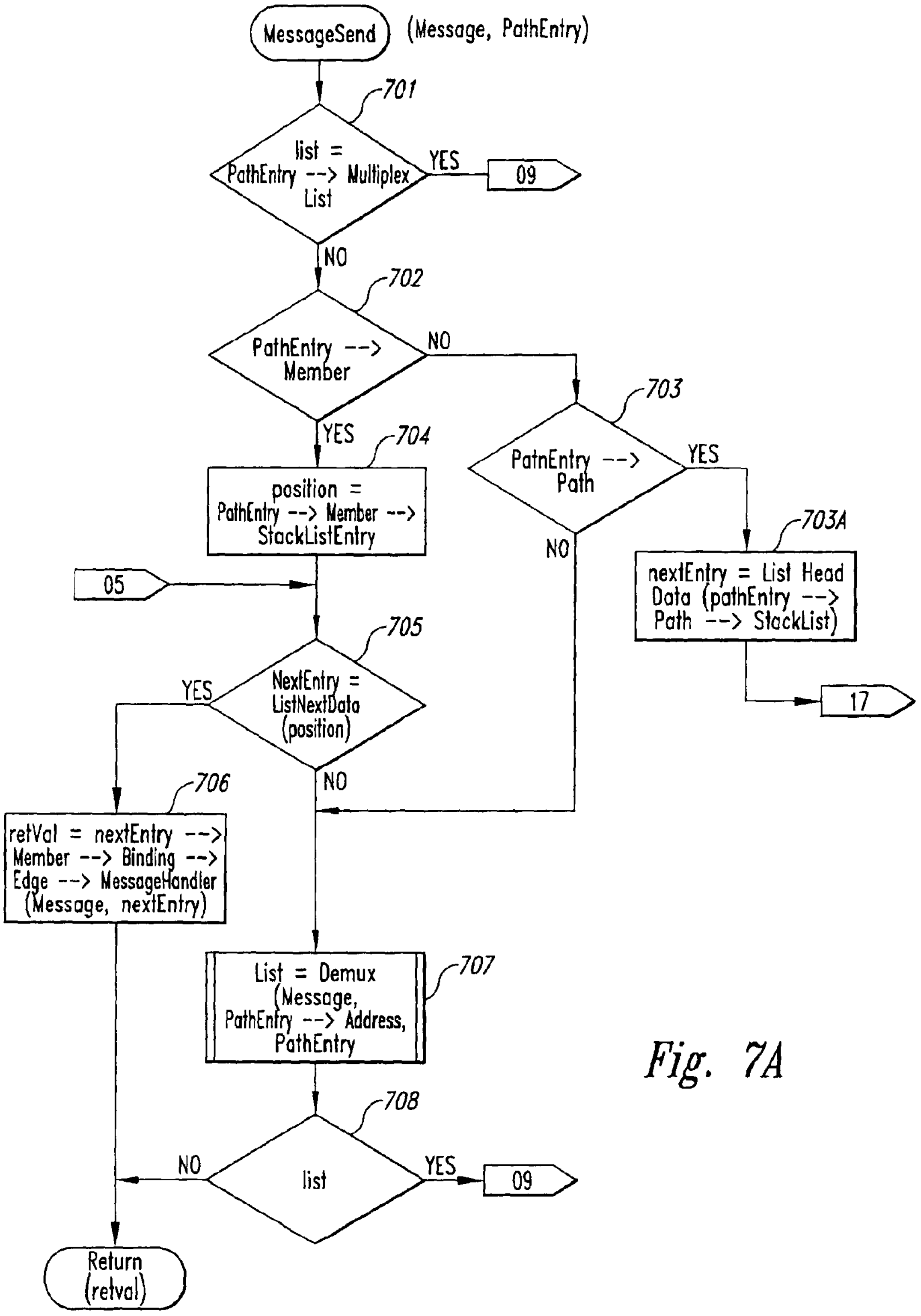

MessageSend
(Message, PathEntry)
701
list = PathEntry --> Multiplex List
YES
09
NO
702
PathEntry --> Member
NO
703
PatnEntry --> Path
YES
703A
nextEntry = List Head Data (pathEntry --> Path --> StackList)
17
NO
YES
704
position = PathEntry --> Member --> StackListEntry
05
705
NextEntry = ListNextData (position)
YES
706
retVal = nextEntry --> Member --> Binding --> Edge --> MessageHandler (Message, nextEntry)
NO
707
List = Demux (Message, PathEntry --> Address, PathEntry
708
list
NO
YES
09
Return (retval)

Fig. 7A

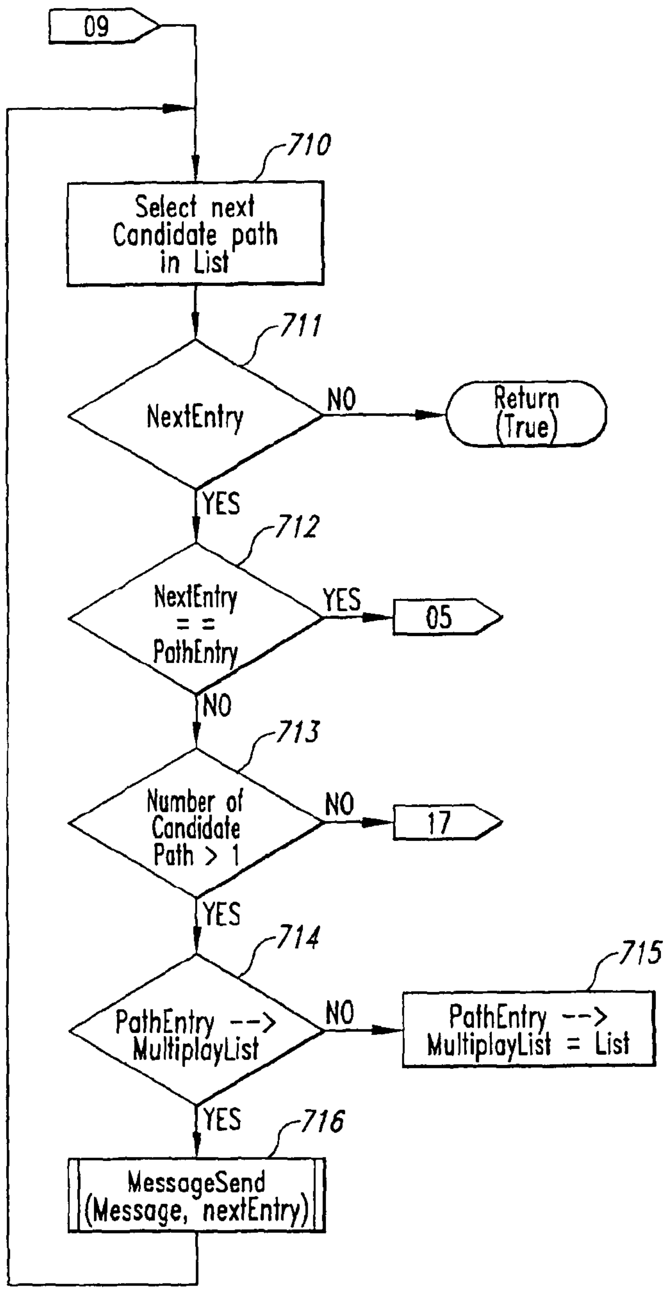

09
710
Select next Candidate path in List
711
NextEntry
NO
Return (True)
YES
712
NextEntry = = PathEntry
YES
05
NO
713
Number of Candidate Path > 1
NO
17
YES
714
PathEntry --> MultiplayList
NO
715
PathEntry --> MultiplayList = List
YES
716
MessageSend (Message, nextEntry)

Fig. 7B

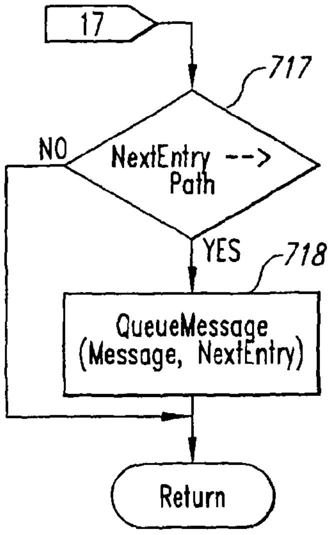

17
717
NO
NextEntry --> Path
YES
718
QueueMessage (Message, NextEntry)
Return

Fig. 7C

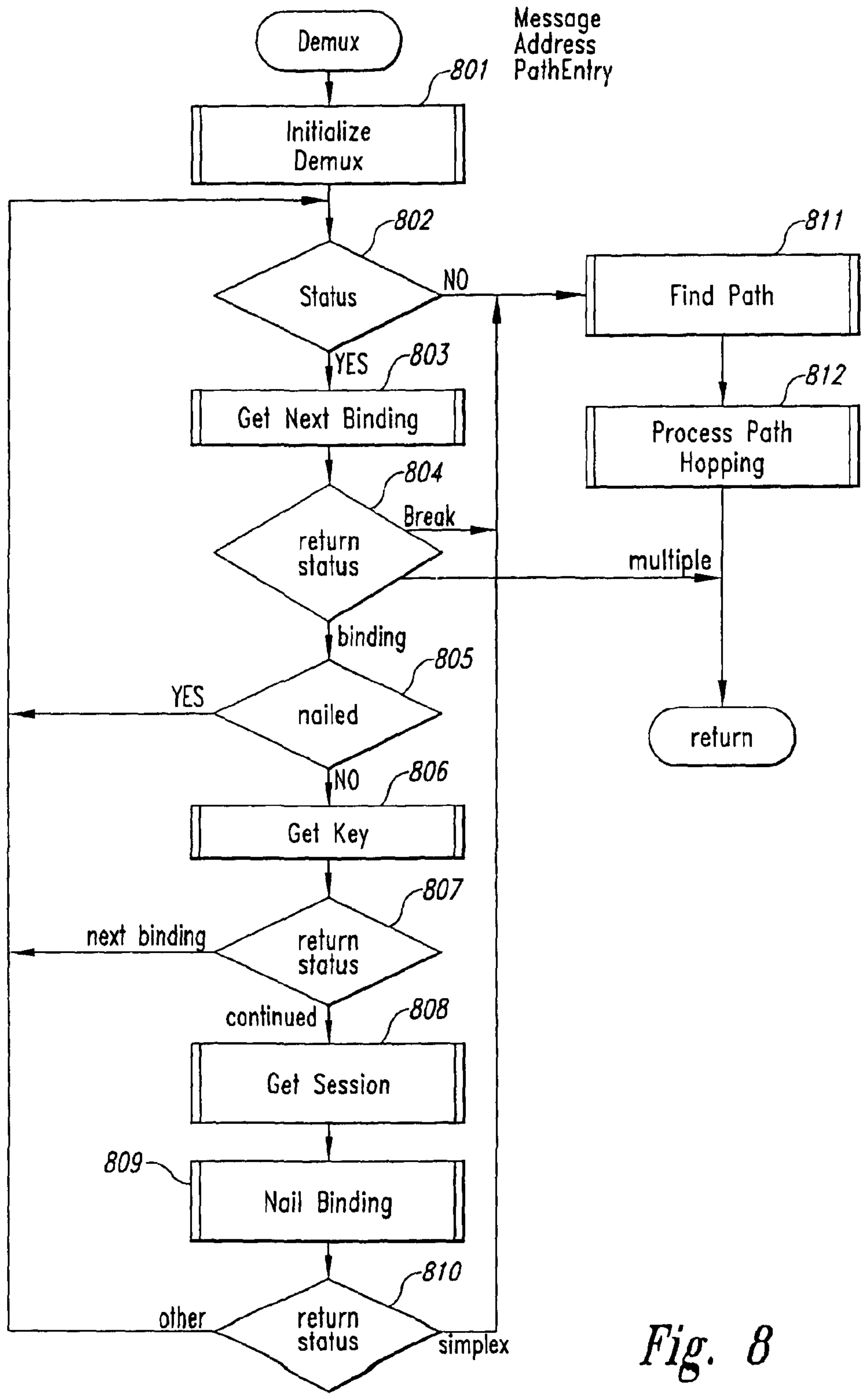

Demux
Message
Address
PathEntry
801
Initialize Demux
802
Status
NO
YES
811
Find Path
803
Get Next Binding
812
Process Path Hopping
804
return status
Break
multiple
binding
805
YES
nailed
return
NO
806
Get Key
807
next binding
return status
continued
808
Get Session
809
Nail Binding
810
other
return status
simplex

Fig. 8

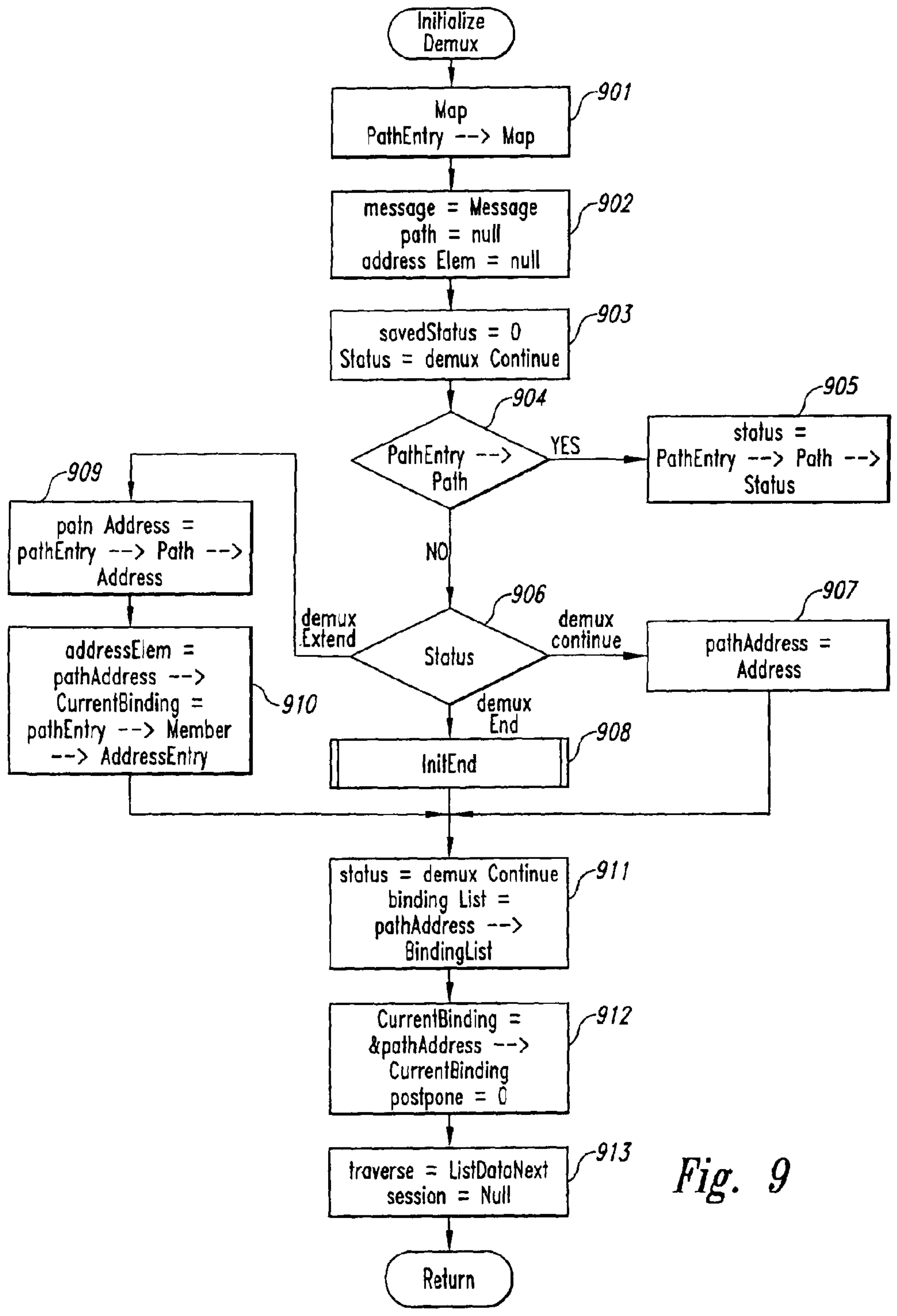

Initialize Demux
901
Map PathEntry --> Map
902
message = Message
path = null
address Elem = null
903
savedStatus = 0
Status = demux Continue
904
PathEntry --> Path
YES
905
status = PathEntry --> Path --> Status
NO
909
patn Address = pathEntry --> Path --> Address
demux Extend
906
Status
demux continue
907
pathAddress = Address
addressElem = pathAddress --> CurrentBinding = pathEntry --> Member --> AddressEntry
910
demux End
908
InitEnd
911
status = demux Continue
binding List = pathAddress --> BindingList
912
CurrentBinding = &pathAddress --> CurrentBinding
postpone = 0
913
traverse = ListDataNext
session = Null
Return

Fig. 9

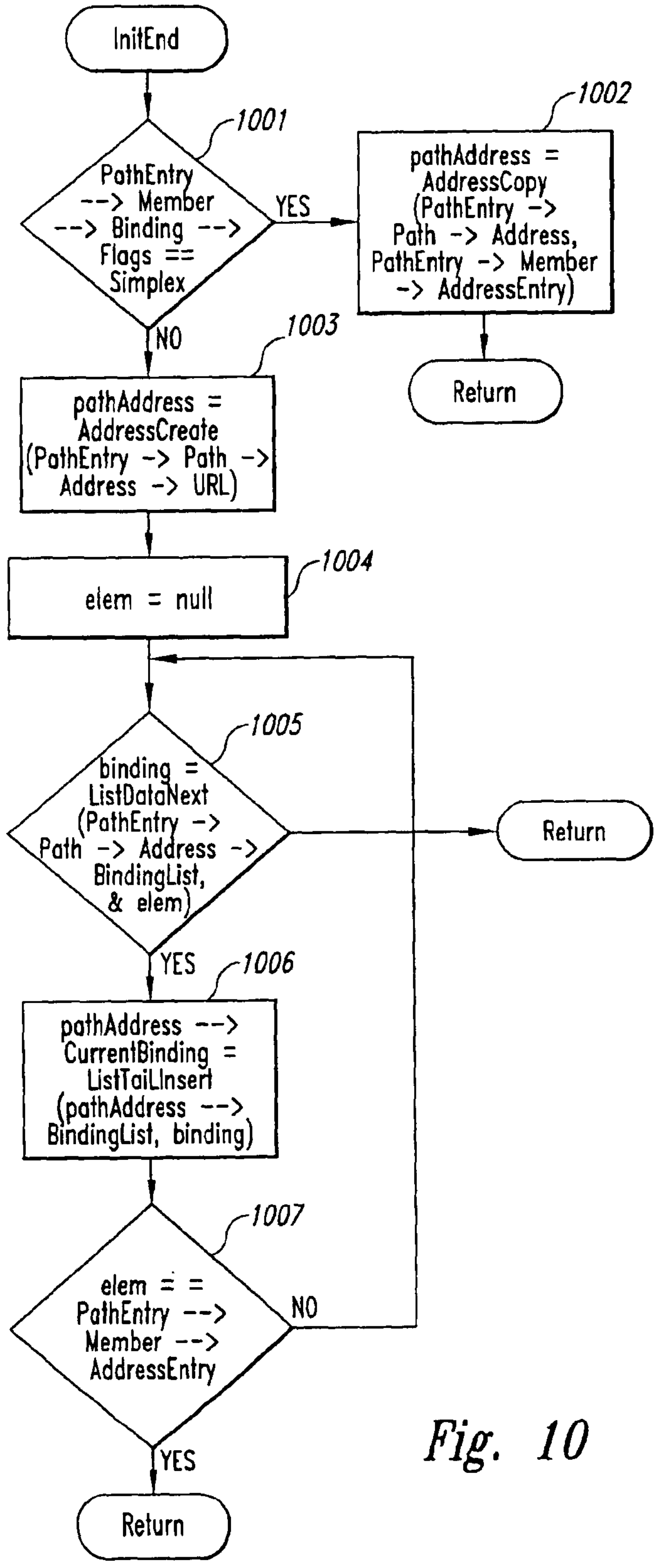

InitEnd
1001
PathEntry --> Member --> Binding --> Flags == Simplex
YES
1002
pathAddress = AddressCopy (PathEntry -> Path -> Address, PathEntry -> Member -> AddressEntry)
Return
NO
1003
pathAddress = AddressCreate (PathEntry -> Path -> Address -> URL)
1004
elem = null
1005
binding = ListDataNext (PathEntry -> Path -> Address -> BindingList, & elem)
Return
YES
1006
pathAddress --> CurrentBinding = ListTailInsert (pathAddress --> BindingList, binding)
1007
elem = = PathEntry --> Member --> AddressEntry
NO
YES
Return

Fig. 10

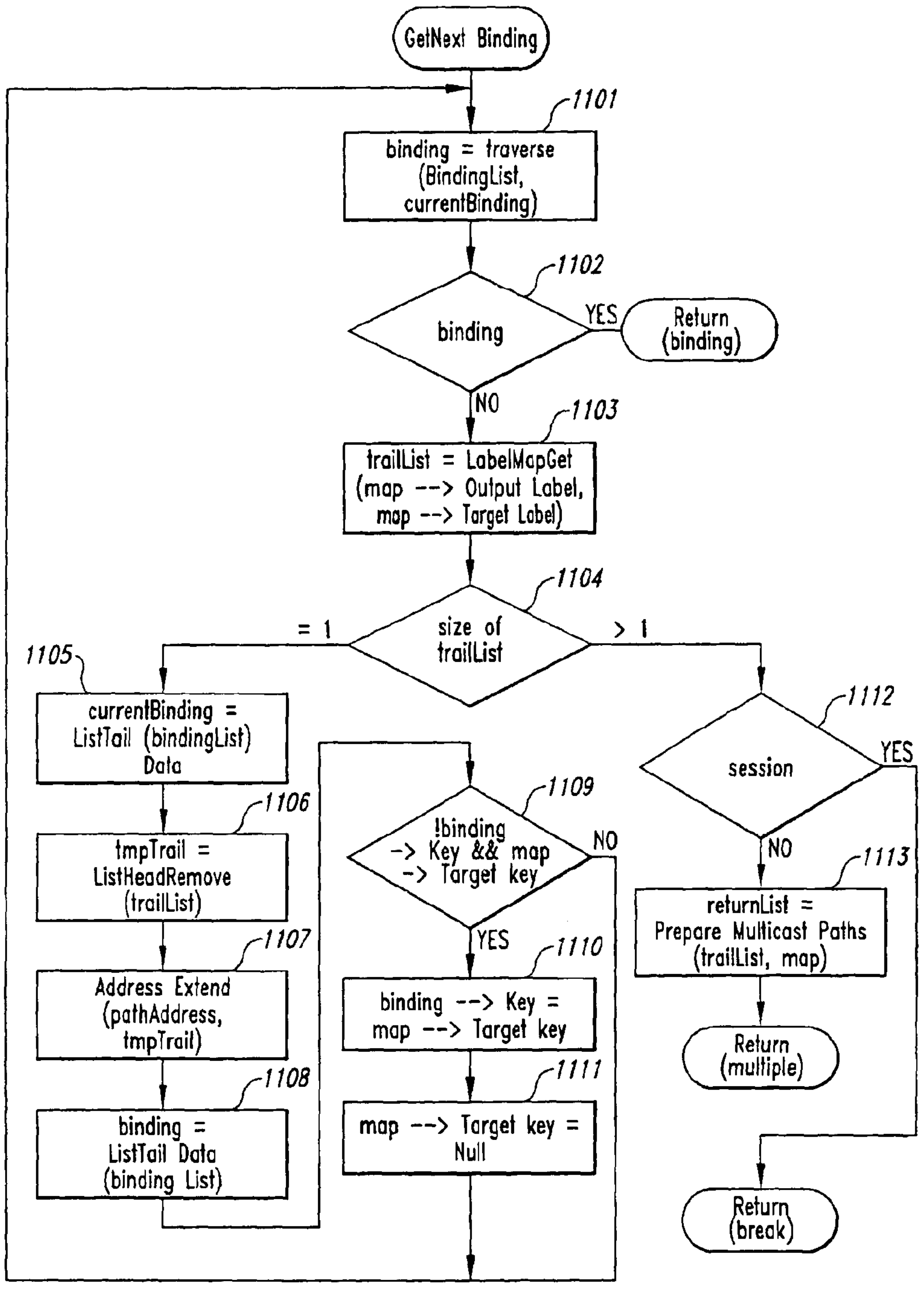
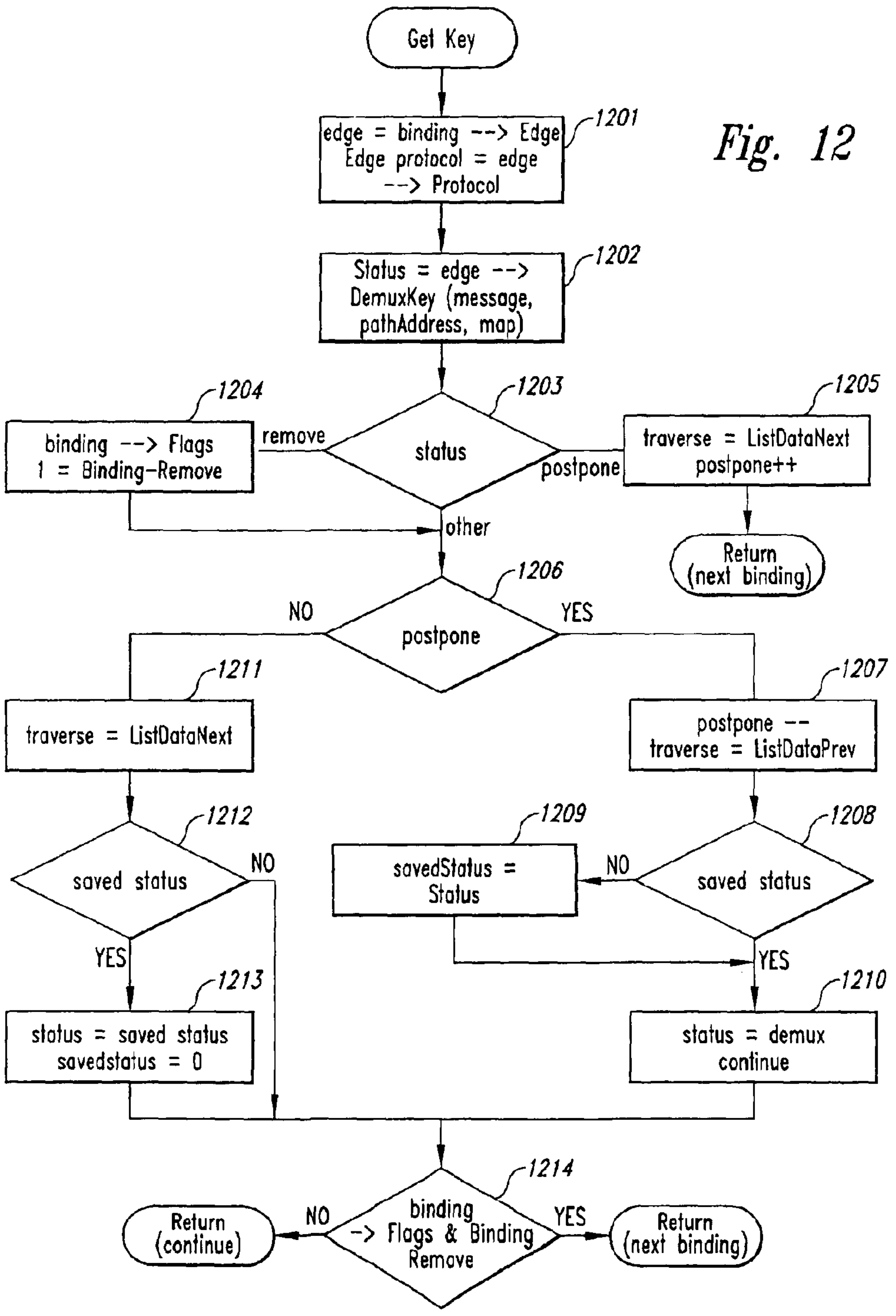

GetNext Binding
1101
binding = traverse (BindingList, currentBinding)
1102
binding
YES
Return (binding)
NO
1103
trailList = LabelMapGet (map --> Output Label, map --> Target Label)
1104
size of trailList
= 1
> 1
1105
currentBinding = ListTail (bindingList) Data
1106
tmpTrail = ListHeadRemove (trailList)
1107
Address Extend (pathAddress, tmpTrail)
1108
binding = ListTail Data (binding List)
1109
!binding -> Key && map -> Target key
NO
YES
1110
binding --> Key = map --> Target key
1111
map --> Target key = Null
1112
session
YES
NO
1113
returnList = Prepare Multicast Paths (trailList, map)
Return (multiple)
Return (break)

Fig. 11



Get Key
edge = binding --> Edge Edge protocol = edge --> Protocol
1201
Status = edge --> DemuxKey (message, pathAddress, map)
1202
status
1203
remove
binding --> Flags 1 = Binding-Remove
1204
postpone
traverse = ListDataNext postpone++
1205
Return (next binding)
other
postpone
1206
NO
YES
traverse = ListDataNext
1211
postpone -- traverse = ListDataPrev
1207
saved status
1212
NO
YES
status = saved status savedstatus = 0
1213
saved status
1208
NO
savedStatus = Status
1209
YES
status = demux continue
1210
binding -> Flags & Binding Remove
1214
NO
Return (continue)
YES
Return (next binding)

Fig. 12

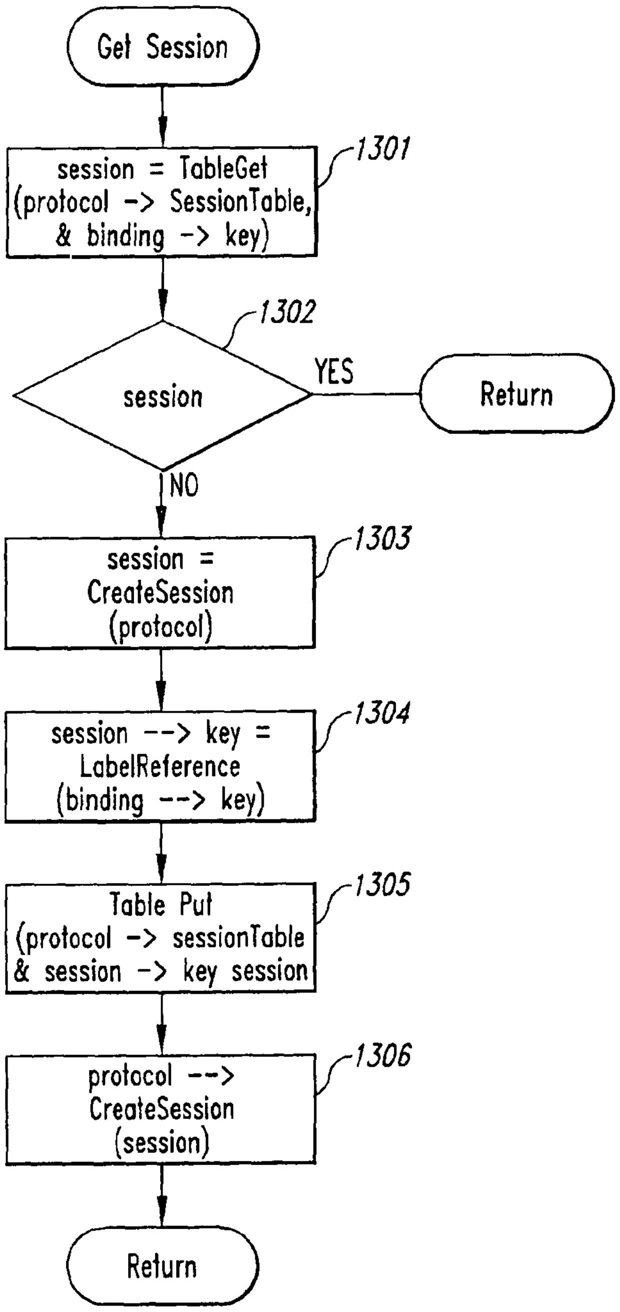

Get Session
session = TableGet (protocol -> SessionTable, & binding -> key)
1301
session
1302
YES
Return
NO
session = CreateSession (protocol)
1303
session --> key = LabelReference (binding --> key)
1304
Table Put (protocol -> sessionTable & session -> key session
1305
protocol --> CreateSession (session)
1306
Return

Fig. 13

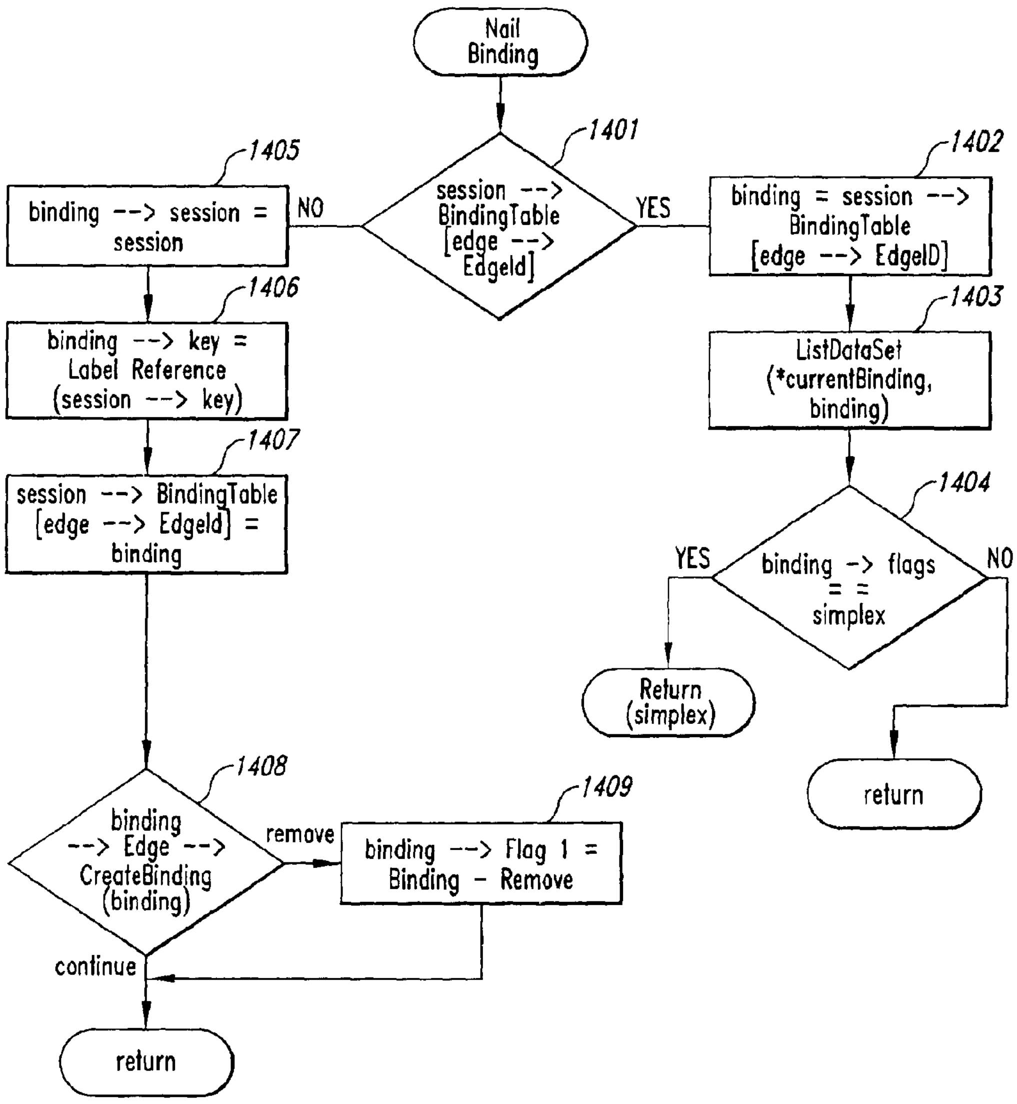

Nail Binding
1401
session --> BindingTable [edge --> EdgeId]
YES
NO
1402
binding = session --> BindingTable [edge --> EdgeID]
1403
ListDataSet (*currentBinding, binding)
1404
binding -> flags = = simplex
YES
NO
Return (simplex)
return
1405
binding --> session = session
1406
binding --> key = Label Reference (session --> key)
1407
session --> BindingTable [edge --> EdgeId] = binding
1408
binding --> Edge --> CreateBinding (binding)
remove
1409
binding --> Flag 1 = Binding - Remove
continue
return

Fig. 14

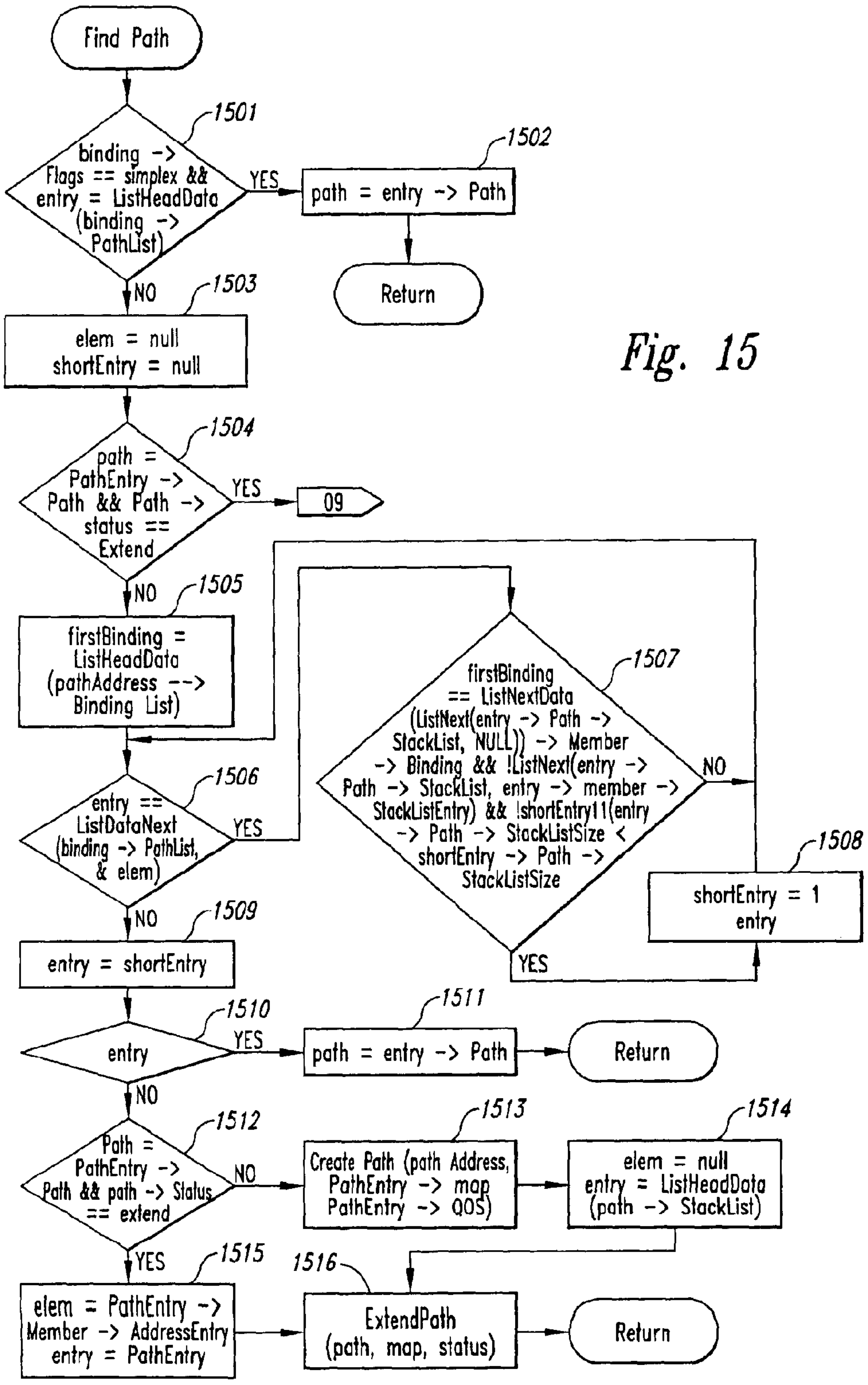

Find Path
1501
binding -> Flags == simplex && entry = ListHeadData (binding -> PathList)
YES
1502
path = entry -> Path
Return
NO
1503
elem = null
shortEntry = null
1504
path = PathEntry -> Path && Path -> status == Extend
YES
09
NO
1505
firstBinding = ListHeadData (pathAddress --> Binding List)
1506
entry == ListDataNext (binding -> PathList, & elem)
YES
1507
firstBinding == ListNextData (ListNext(entry -> Path -> StackList, NULL)) -> Member -> Binding && !ListNext(entry -> Path -> StackList, entry -> member -> StackListEntry) && !shortEntry11(entry -> Path -> StackListSize < shortEntry -> Path -> StackListSize
NO
YES
1508
shortEntry = 1 entry
NO
1509
entry = shortEntry
1510
entry
YES
1511
path = entry -> Path
Return
NO
1512
Path = PathEntry -> Path && path -> Status == extend
NO
1513
Create Path (path Address, PathEntry -> map PathEntry -> QOS)
1514
elem = null
entry = ListHeadData (path -> StackList)
YES
1515
elem = PathEntry -> Member -> AddressEntry
entry = PathEntry
1516
ExtendPath (path, map, status)
Return

Fig. 15

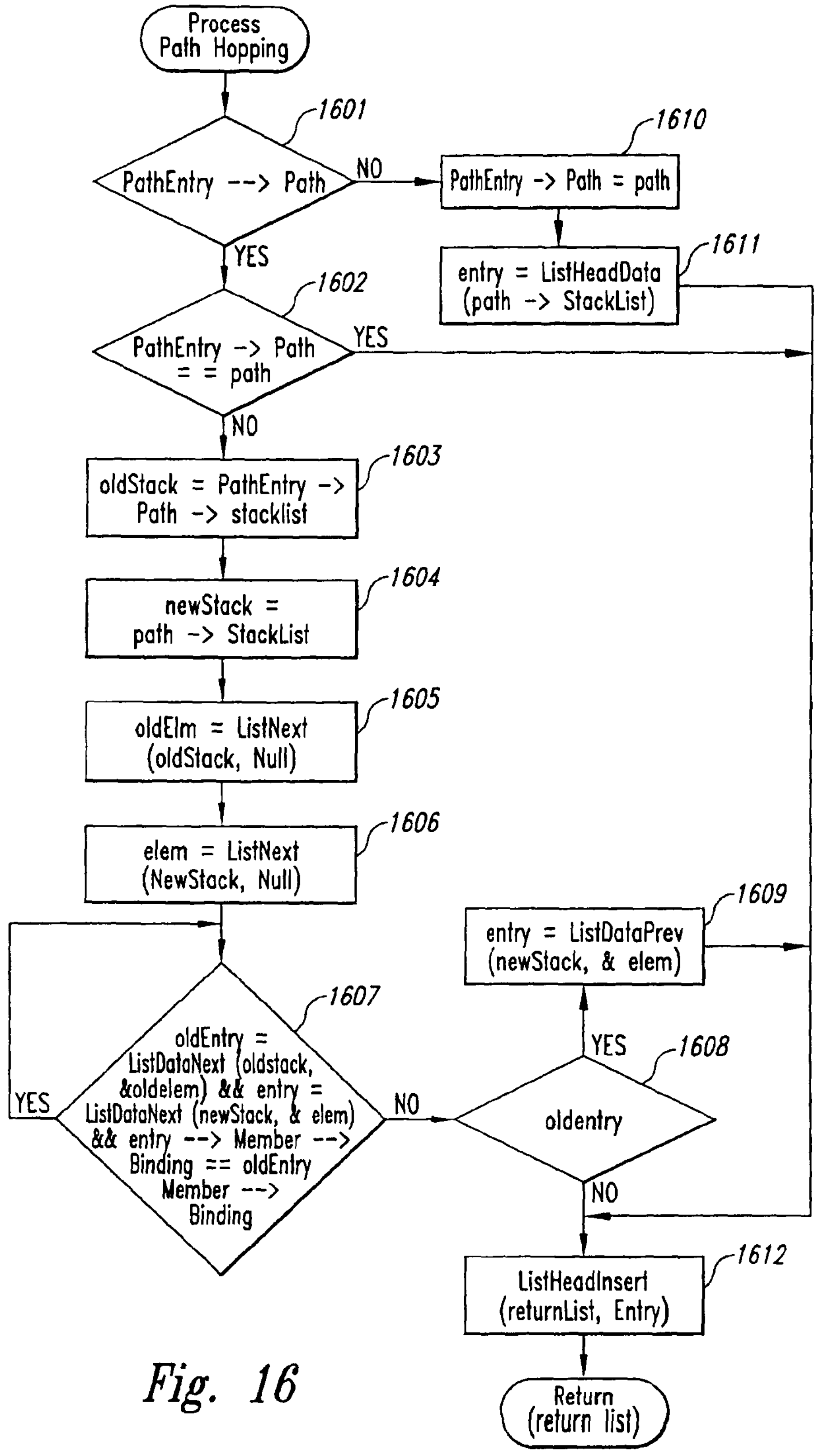

Process Path Hopping
1601
PathEntry --> Path
NO
YES
1610
PathEntry -> Path = path
1611
entry = ListHeadData (path -> StackList)
1602
PathEntry -> Path = = path
YES
NO
1603
oldStack = PathEntry -> Path -> stacklist
1604
newStack = path -> StackList
1605
oldElm = ListNext (oldStack, Null)
1606
elem = ListNext (NewStack, Null)
1609
entry = ListDataPrev (newStack, & elem)
1607
oldEntry = ListDataNext (oldstack, &oldelem) && entry = ListDataNext (newStack, & elem) && entry --> Member --> Binding == oldEntry Member --> Binding
YES
NO
YES
1608
oldentry
NO
1612
ListHeadInsert (returnList, Entry)
Return (return list)

Fig. 16

METHOD AND SYSTEM FOR DATA DEMULTIPLEXING

CROSS REFERENCES TO RELATED APPLICATIONS

The present application is a continuation of U.S. application Ser. No. 13/236,090, filed Sep. 19, 2011, which is a continuation of U.S. application Ser. No. 10/636,314, filed Aug. 6, 2003 (now U.S. Pat. No. 8,055,786), which is a continuation of U.S. application Ser. No. 09/474,664, filed Dec. 29, 1999 (now U.S. Pat. No. 6,629,163); the disclosures of each of the above-referenced applications are incorporated by reference herein in their entireties.

TECHNICAL FIELD

The present invention relates generally to a computer system for data demultiplexing.

BACKGROUND

Computer systems, which are becoming increasingly pervasive, generate data in a wide variety of formats. The Internet is an example of interconnected computer systems that generate data in many different formats. Indeed, when data is generated on one computer system and is transmitted to another computer system to be displayed, the data may be converted in many different intermediate formats before it is eventually displayed. For example, the generating computer system may initially store the data in a bitmap format. To send the data to another computer system, the computer system may first compress the bitmap data and then encrypt the compressed data. The computer system may then convert that compressed data into a TCP format and then into an IP format. The IP formatted data may be converted into a transmission format, such as an ethernet format. The data in the transmission format is then sent to a receiving computer system. The receiving computer system would need to perform each of these conversions in reverse order to convert the data in the bitmap format. In addition, the receiving computer system may need to convert the bitmap data into a format that is appropriate for rendering on output device.

In order to process data in such a wide variety of formats, both sending and receiving computer systems need to have many conversion routines available to support the various formats. These computer systems typically use predefined configuration information to load the correct combination of conversion routines for processing data. These computer systems also use a process-oriented approach when processing data with these conversion routines. When using a process-oriented approach, a computer system may create a separate process for each conversion that needs to take place. A computer system in certain situations, however, can be expected to receive data and to provide data in many different formats that may not be known until the data is received. The overhead of statically providing each possible series of conversion routines is very high. For example, a computer system that serves as a central controller for data received within a home would be expected to process data received via telephone lines, cable TV lines, and satellite connections in many different formats. The central controller would be expected to output the data to computer displays, television displays, entertainment centers, speakers, recording devices, and so on in many different formats. Moreover, since the various conversion routines may be developed by different organizations, it may not be easy to identify that the output format of one conversion routine is compatible with the input format of another conversion routine.

It would be desirable to have a technique for dynamically identifying a series of conversion routines for processing data. In addition, it would be desirable to have a technique in which the output format of one conversion routine can be identified as being compatible with the input format of another conversion routine. It would also be desirable to store the identification of a series of conversion routines so that the series can be quickly identified when data is received.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a block diagram illustrating example processing of a message by the conversion system.

FIG. **2** is a block diagram illustrating a sequence of edges.

FIG. **3** is a block diagram illustrating components of the conversion system in one embodiment.

FIG. **4** is a block diagram illustrating example path data structures in one embodiment.

FIG. **5** is a block diagram that illustrates the interrelationship of the data structures of a path.

FIG. **6** is a block diagram that illustrates the interrelationship of the data structures associated with a session.

FIGS. **7**A, **7**B, and **7**C comprise a flow diagram illustrating the processing of the message send routine.

FIG. **8** is a flow diagram of the demux routine.

FIG. **9** is a flow diagram of the initialize demux routine.

FIG. **10** is a flow diagram of the init end routine.

FIG. **11** is a flow diagram of a routine to get the next binding.

FIG. **12** is a flow diagram of the get key routine.

FIG. **13** is a flow diagram of the get session routine.

FIG. **14** is a flow diagram of the nail binding routine.

FIG. **15** is a flow diagram of the find path routine.

FIG. **16** is a flow diagram of the process of path hopping routine.

DETAILED DESCRIPTION

A method and system for converting a message that may contain multiple packets from an source format into a target format. When a packet of a message is received, the conversion system in one embodiment searches for and identifies a sequence of conversion routines (or more generally message handlers) for processing the packets of the message by comparing the input and output formats of the conversion routines. (A message is a collection of data that is related in some way, such as stream of video or audio data or an email message.) The identified sequence of conversion routines is used to convert the message from the source format to the target format using various intermediate formats. The conversion system then queues the packet for processing by the identified sequence of conversion routines. The conversion system stores the identified sequence so that the sequence can be quickly found (without searching) when the next packet in the message is received. When subsequent packets of the message are received, the conversion system identifies the sequence and queues the packets for pressing by the sequence. Because the conversion system receives multiple messages with different source and target formats and identifies a sequence of conversion routines for each message, the conversion systems effectively “demultiplexes” the messages. That is, the conversion system demultiplexes the messages by receiving the message, identifying the sequence of conversion routines, and controlling the processing of each message by the identified sequence. Moreover, since the conversion routines may need to retain state information between the receipt of one packet of a message and the next packet of that message, the conversion system maintains state information as an instance or session of the conversion routine. The conversion system routes all packets for a message through the same session of each conversion routine so that the same state or instance information can be used by all packets of the message. A sequence of sessions of conversion routines is referred to as a "path." In one embodiment, each path has a path thread associated with it for processing of each packet destined for that path.

In one embodiment, the packets of the messages are initially received by "drivers," such as an Ethernet driver. When a driver receives a packet, it forwards the packet to a forwarding component of the conversion system. The forwarding component is responsible for identifying the session of the conversion routine that should next process the packet and invoking that conversion routine. When invoked by a driver, the forwarding component may use a demultiplexing ("demux") component to identify the session of the first conversion routine of the path that is to process the packet and then queues the packet for processing by the path. A path thread is associated with each path. Each path thread is responsible for retrieving packets from the queue of its path and forwarding the packets to the forwarding component. When the forwarding component is invoked by a path thread, it initially invokes the first conversion routine in the path. That conversion routine processes the packet and forwards the processed packet to the forwarding component, which then invokes the second conversion routine in the path. The process of invoking the conversion routines and forwarding the processed packet to the next conversion routine continues until the last conversion routine in the path is invoked. A conversion routine may defer invocation of the forwarding component until it aggregates multiple packets or may invoke the forwarding component multiple times for a packet once for each sub-packet.

The forwarding component identifies the next conversion routine in the path using the demux component and stores that identification so that the forwarding component can quickly identify the conversion routine when subsequent packets of the same message are received. The demux component, searches for the conversion routine and session that is to next process a packet. The demux component then stores the identification of the session and conversion routine as part of a path data structure so that the conversion system does not need to search for the session and conversion routine when requested to demultiplex subsequent packets of the same message. When searching for the next conversion routine, the demux component invokes a label map get component that identifies the next conversion routine. Once the conversion routine is found, the demux component identifies the session associated with that message by, in one embodiment, invoking code associated with the conversion routine. In general, the code of the conversion routine determines what session should be associated with a message. In certain situations, multiple messages may share the same session. The demux component then extends the path for processing that packet to include that session and conversion routine. The sessions are identified so that each packet is associated with the appropriate state information. The dynamic identification of conversion routines is described in U.S. patent application Ser. No. 11,933,093, filed on Oct. 31, 2007 (now U.S. Pat. No. 7,730, 211), entitled "Method and System for Generating a Mapping Between Types of Data," which is hereby incorporated by reference.

FIG. **1** is a block diagram illustrating example processing of a message by the conversion system. The driver **101** receives the packets of the message from a network. The driver performs any appropriate processing of the packet and invokes a message send routine passing the processed packet along with a reference path entry **150**. The message send routine is an embodiment of the forwarding component. A path is represented by a series of path entries, which are represented by triangles. Each member path entry represents a session and conversion routine of the path, and a reference path entry represents the overall path. The passed reference path entry **150** indicates to the message send routine that it is being invoked by a driver. The message send routine invokes the demux routine **102** to search for and identify the path of sessions that is to process the packet. The demux routine may in tum invoke the label map get routine **104** to identify a sequence of conversion routines for processing the packet. In this example, the label map get routine identifies the first three conversion routines, and the demux routine creates the member path entries **151**, **152**, **153** of the path for these conversion routines. Each path entry identifies a session for a conversion routine, and the sequence of path entries **151**-**155** identifies a path. The message send routine then queues the packet on the queue **149** for the path that is to process the packets of the message. The path thread **105** for the path retrieves the packet from the queue and invokes the message send routine **106** passing the packet and an indication of the path. The message send routine determines that the next session and conversion routine as indicated by path entry **151** has already been found. The message send routine then invokes the instance of the conversion routine for the session. The conversion routine processes the packet and then invokes the message send routine **107**. This processing continues until the message send routine invokes the demux routine **110** after the packet is processed by the conversion routine represented by path entry **153**. The demux routine examines the path and determines that it has no more path entries. The demux routine then invokes the label map get routine **111** to identify the conversion routines for further processing of the packet. When the conversion routines are identified, the demux routine adds path entries **154**, **155** to the path. The messages send routine invokes the conversion routine associated with path entry **154**. Eventually, the conversion routine associated with path entry **155** performs the final processing for the path.

The label map get routine identifies a sequence of "edges" for converting data in one format into another format. Each edge corresponds to a conversion routine for converting data from one format to another. Each edge is part of a "protocol" (or more generally a component) that may include multiple related edges. For example, a protocol may have edges that each convert data in one format into several different formats. Each edge has an input format and an output format. The label map get routine identifies a sequence of edges such that the output format of each edge is compatible with the input format of another edge in the sequence, except for the input format of the first edge in the sequence and the output format of the last edge in the sequence. FIG. **2** is a block diagram illustrating a sequence of edges. Protocol PI includes an edge for converting format D1 to format D2 and an edge for converting format D1 to format D3; protocol P2 includes an edge for converting format D2 to format D5, and so on. A 30 sequence for converting format D 1 to format D 15 is shown by the curved lines and is defined by the address "P1:I, P2:1, P3:2, P4:7." When a packet of data in format D I is processed by this sequence, it is converted to format DIS. During the process, the packet of data is sequentially converted to format D2, D5, and D13. The output format of protocol P2, edge 1 (i.e., P2:1) is format D5, but the input format of P3:2 is format D10. The label map get routine uses an aliasing mechanism by which two formats, such as D5 and D10 are identified as being compatible. The use of aliasing allows different names of the same format or compatible formats to be correlated.

FIG. **3** is a block diagram illustrating components of the conversion system in one embodiment. The conversion system **300** can operate on a computer system with a central processing unit **301**, I/O devices **302**, and memory **303**. The **110** devices may include an Internet connection, a connection to various output devices such as a television, and a connection to various input devices such as a television receiver. The media mapping system may be stored as instructions on a computer-readable medium, such as a disk drive, memory, or data transmission medium. The data structures of the media mapping system may also be stored on a computer-readable medium. The conversion system includes drivers **304**, a forwarding component **305**, a demux component **306**, a label map get component **307**, path data structures **308**, conversion routines **309**, and instance data **310**. Each driver receives data in a source format and forwards the data to the forwarding component. The forwarding component identifies the next conversion routine in the path and invokes that conversion routine to process a packet. The forwarding component may invoke the demux component to search for the next conversion routine and add that conversion routine to the path. The demux component may invoke the label map get component to identify the next conversion routine to process the packet. The demux component stores information defining the paths in the path structures. The conversion routines store their state information in the instance data.

FIG. **4** is a block diagram illustrating example path data structures in one embodiment. The demux component identifies a sequence of "edges" for converting data in one format into another format by invoking the label map get component. Each edge corresponds to a conversion routine for converting data from one format to another. As discussed above, each edge is part of a "protocol" that may include multiple related edges. For example, a protocol may have edges that each convert data in one format into several different formats. Each edge has as an input format ("input label") and an output format ("output label"). Each rectangle represents a session **410**, **420**, **430**, **440**, **450** for a protocol. A session corresponds to an instance of a protocol. That is, the session includes the protocol and state information associated with that instance of the protocol. Session **410** corresponds to a session for an Ethernet protocol; session **420** corresponds to a session for an IP protocol; and sessions **430**, **440**, **450** correspond to sessions for a TCP protocol. FIG. **4** illustrates three paths **461**, **462**, **463**. Each path includes edges **411**, **421**, **431**. The paths share the same Ethernet session **410** and IP session **420**, but each path has a unique TCP session **430**, **440**, **450**. Thus, path **461** includes sessions **410**, **420**, and **430**; path **462** includes sessions **410**, **420**, and **440**; and path **463** includes sessions **410**, **420**, and **450**. The conversion system represents each path by a sequence of path entry structures. Each path entry structure is represented by a triangle. Thus, path **461** is represented by path entries **415**, **425**, and **433**. The conversion system represents the path entries of a path by a stack list. Each path also has a queue **471**, **472**, **473** associated with it. Each queue stores the messages that are to be processed by the conversion routines of the edges of the path. Each session includes a binding **412**, **422**, **432**, **442**, **452** that is represented by an oblong shape adjacent to the corresponding edge. A binding for an edge of a session represents those paths that include the edge. The binding **412** indicates that three paths are bound (or "nailed") to edge **411** of the Ethernet session **410**. The conversion system uses a path list to track the paths that are bound to a binding. The path list of binding **412** identifies path entries **413**, **414**, and **415**.

FIG. **5** is a block diagram that illustrates the interrelationship of the data structures of a path. Each path has a corresponding path structure **501** that contains status information and pointers to a message queue structure **502**, a stack list structure **503**, and a path address structure **504**. The status of a path can be extend, continue, or end. Each message handler returns a status for the path. The status of extend means that additional path entries should be added to the path. The status of end means that this path should end at this point and subsequent processing should continue at a new path. The status of continue means that the protocol does not care how the path is handled. In one embodiment, when a path has a status of continue, the system creates a copy of the path and extends the copy. The message queue structure identifies the messages (or packets of a message) that are queued up for processing by the path and identifies the path entry at where the processing should start. The stack list structure contains a list of pointers to the path entry structures **505** that comprise the path. Each path entry structure contains a pointer to the corresponding path data structure, a pointer to a map structure **507**, a pointer to a multiplex list **508**, a pointer to the corresponding path address structure, and a pointer to a member structure **509**. A map structure identifies the output label of the edge of the path entry and optionally a target label and a target key. A target key identifies the session associated with the protocol that converts the packet to the target label. (The terms "media," "label," and "format" are used interchangeably to refer to the output of a protocol.) The multiplex list is used during the demux process to track possible next edges when a path is being identified as having more than one next edge. The member structure indicates that the path entry represents an edge of a path and contains a pointer to a binding structure to which the path entry is associated (or "nailed"), a stack list entry is the position of the path entry within the associated stack list, a path list entry is the position of the path entry within the associated path list of a binding and an address entry is the position of the binding within the associated path address. A path address of a path identifies the bindings to which the path entries are bound. The path address structure contains a URL for the path, the name of the path identified by the address, a pointer to a binding list structure **506**, and the identification of the current binding within the binding list. The URL (e.g., "protocol://tcp(0)/ip(0)/eth(0)") identifies conversion routines (e.g., protocols and edges) of a path in a human-readable format. The URL (universal resource locator) includes a type field (e.g., "protocol") followed by a sequence of items (e.g.,"tcp(0)"). The type field specifies the format of the following information in the URL, that specifies that the type field is followed by a sequence of items. Each item identifies a protocol and an edge (e.g., the protocol is "tcp" and the edge is "0"). In one embodiment, the items of a URL may also contain an identifier of state information that is to be used when processing a message. These URLs can be used to illustrate to a user various paths that are available for processing a message. The current binding is the last binding in the path as the path is being built. The binding list structure contains a list of pointers to the binding structures associated with the path. Each binding structure **510** contains a pointer to a session structure, a pointer to an edge structure, a key, a path list structure, and a list of active paths through the binding. The key identifies the state information for a session of a protocol. A path list structure contains pointers to the path entry structures associated with the binding.

FIG. **6** is a block diagram that illustrates the interrelationship of the data structures associated with a session. A session structure **601** contains the context for the session, a pointer to a protocol structure for the session, a pointer to a binding table structure **602** for the bindings associated with the session, and the key. The binding table structure contains a list of pointers to the binding structures **510** for the session. The binding structure is described above with reference to FIG. **5**. The path list structure **603** of the binding structure contains a list of pointers to path entry structures **505**. The path entry structures are described with reference to FIG. **5**.

FIGS. **7**A, **7**B, and **7**C comprise a flow diagram illustrating the processing of the message send routine. The message send routine is passed a message along with the path entry associated with the session that last processed the message. The message send routine invokes the message handler of the next edge in the path or queues the message for processing by a path. The message handler invokes the demux routine to identify the next path entry of the path. When a driver receives a message, it invokes the message send routine passing a reference path entry. The message send routine examines the passed path entry to determine (1) whether multiple paths branch from the path of the passed path entry, (2) whether the passed path entry is a reference with an associated path, or (3) whether the passed path entry is a member with a next path entry. If multiple paths branch from the path of the passed path entry, then the routine recursively invokes the message send routine for each path. If the path entry is a reference with an associated path, then the driver previously invoked the message send routine, which associated a path with the reference path entry, and the routine places the message on the queue for the path. If the passed path entry is a member with a next path entry, then the routine invokes the message handler (i.e., conversion routine of the edge) associated with the next path entry. If the passed path entry is a reference without an associated path or is a member without a next path entry, then the routine invokes the demux routine to identify the next path entry. The routine then recursively invokes the messages send routine passing that next path entry. In decision block **701**, if the passed path entry has a multiplex list, then the path branches off into multiple paths and the routine continues at block **709**, else the routine continues at block **702**. A packet may be processed by several different paths. For example, if a certain message is directed to two different output devices, then the message is processed by two different paths. Also, a message may need to be processed by multiple partial paths when searching for a complete path. In decision block **702**, if the passed path entry is a member, then either the next path entry indicates a nailed binding or the path needs to be extended and the routine continues at block **704**, else the routine continues at block **703**. A nailed binding is a binding (e.g., edge and protocol) is associated with a session. In decision block **703**, the passed path entry is a reference and if the passed path entry has an associated path, then the routine can queue the message for the associated path and the routine continues at block **703**A, else the routine needs to identify a path and the routine continues at block **707**. In block **703**A, the routine sets the entry to the first path entry in the path and continues at block **717**. In block **704**, the routine sets the variable position to the stack list entry of the passed path entry. In decision block **705**, the routine sets the variable next entry to the next path entry in the path. If there is a next entry in the path, then the next session and edge of the protocol have been identified and the routine continues at block **706**, else the routine continues at block **707**. In block **706**, the routine passes the message to the message handler of the edge associated with the next entry and then returns. In block **706**, the routine invokes the demux routine passing the passed message, the address of the passed path entry, and the passed path entry. The demux routine returns a list of candidate paths for processing of the message. In decision block **708**, if at least one candidate path is returned, then the routine continues at block **709**, else the routine returns.

Blocks **709-716** illustrate the processing of a list of candidate paths that extend from the passed path entry. In blocks **710-716**, the routine loops selecting each candidate path and sending the message to be process by each candidate path. In block **710**, the routine sets the next entry to the first path entry of the next candidate path. In decision block **711**, if all the candidate paths have not yet been processed, then the routine continues at block **712**, else the routine returns. In decision block **712**, if the next entry is equal to the passed path entry, then the path is to be extended and the routine continues at block **705**, else the routine continues at block **713**. The candidate paths include a first path entry that is a reference path entry for new paths or that is the last path entry of a path being extended. In decision block **713**, if the number of candidate paths is greater than one, then the routine continues at block **714**, else the routine continues at block **718**. In decision block **714**, if the passed path entry has a multiplex list associated with it, then the routine continues at block **716**, else the routine continues at block **715**. In block **715**, **11** the routine associates the list of candidate path with the multiplex list of the passed path entry and continues at block **716**. In block **716**, the routine sends the message to the next entry by recursively invoking the message send routine. The routine then loops to block **710** to select the next entry associated with the next candidate path.

Blocks **717-718** are performed when the passed path entry is a reference path entry that has a path associated with it. In block **717**, if there is a path associated with the next entry, then the routine continues at block **718**, else the routine returns. In block **718**, the routine queues the message for the path of the next entry and then returns.

FIG. **8** is a flow diagram of the demux routine. This routine is passed the packet (message) that is received, an address structure, and a path entry structure. The demux routine extends a path, creating one if necessary. The routine loops identifying the next binding (edge and protocol) that is to process the message and "nailing" the binding to a session for the message, if not already nailed. After identifying the nailed binding, the routine searches for the shortest path through the nailed binding, creating a path if none exists. In block **801**, the routine invokes the initialize demux routine. In blocks **802-810**, the routine loops identifying a path or portion of a path for processing the passed message. In decision block **802**, if there is a current status, which was returned by the demuxkey routine that was last invoked (e.g., continue, extend, end, or postpone), then the routine continues at block **803**, else the routine continues at block **811**. In block **803**, the routine invokes the get next binding routine. The get next binding routine returns the next binding in the path. The binding is the edge of a protocol. That routine extends the path as appropriate to include the binding. The routine returns a return status of break, binding, or multiple. The return status of binding indicates that the next binding in the path was found by extending the path as appropriate and the routine continues to "nail" the binding to a session as appropriate. The return status of multiple means that multiple trails (e.g., candidate paths) were identified as possible extensions of the path. In a decision block **804**, if the return status is break, then the routine continues at block **811**. If the return status is multiple, then the routine returns. If the return status is binding, then the routine continues at block **805**. In decision block **805**, if the retrieved binding is nailed as indicated by being assigned to a session, then the routine loops to block **802**, else the routine continues at block **806**. In block **806**, the routine invokes the get key routine of the edge associated with the binding. The get key routine creates the key for the session associated with the message. If a key cannot be created until subsequent bindings are processed or because the current binding is to be removed, then the get key routine returns a next binding status, else it returns a continue status. In decision block **807**, if the return status of the get key routine is next binding, then the routine loops to block **802** to get the next binding, else the routine continues at block **808**. In block **808**, the routine invokes the routine get session. The routine get session returns the session associated with the key, creating a new session if necessary. In block **809**, the routine invokes the routine nail binding. The routine nail binding retrieves the binding if one is already nailed to the session. Otherwise, that routine nails the binding to the session. In decision block **810**, if the nail binding routine returns a status of simplex, then the routine continues at block **811** because only one path can use the session, else the routine loops to block **802**. Immediately upon return from the nail binding routine, the routine may invoke a set map routine of the edge passing the session and a map to allow the edge to set its map. In block **811**, the routine invokes the find path routine, which finds the shortest path through the binding list and creates a path if necessary. In block **812**, the routine invokes the process path hopping routine, which determines whether the identified path is part of a different path. Path hopping occurs when, for example, IP fragments are built up along separate paths, but once the fragments are built up they can be processed by the same subsequent path.

FIG. **9** is a flow diagram of the initialize demux routine. This routine is invoked to initialize the local data structures that are used in the demux process and to identify the initial binding. The demux routine finds the shortest path from the initial binding to the final binding. If the current status is demux extend, then the routine is to extend the path of the passed path entry by adding additional path entries. If the current status is demux end, then the demux routine is ending the current path. If the current status is demux continue, then the demux routine is in the process of continuing to extend or in the process of starting a path identified by the passed address. In block **901**, the routine sets the local map structure to the map structure in the passed path entry structure. The map structure identifies the output label, the target label, and the target key. In the block **902**, the routine initializes the local message structure to the passed message structure and initializes the pointers path and address element to null. In block **903**, the routine sets of the variable saved status to 0 and the variable status to demux continue. The variable saved status is used to track the status of the demux process when backtracking to nail a binding whose nail was postponed. In decision block **904**, if the passed path entry is associated with a path, then the routine continues at block **905**, else the routine continues at block **906**. In block **905**, the routine sets the variable status to the status of that path. In block **906**, if the variable status is demux continue, then the routine continues at block **907**. If the variable status is demux end, then the routine continues at block **908**. If the variable status is demux extend, then the routine continues at block **909**. In block **907**, the status is demux continue, and the routine sets the local pointer path address to the passed address and continues at block **911**. In block **908**, the status is demux end, and the routine invokes the init end routine and continues at block **911**. In block **909**, the status is demux extend, and the routine sets the local path address to the address of the path that contains the passed path entry. In block **910**, the routine sets the address element and the current binding of the path address pointed to by the local pointer path address to the address entry of the member structure of the passed path entry. In the block **911**, the routine sets the local variable status to demux continue and sets the local binding list structure to the binding list structure from the local path address structure. In block **912**, the routine sets the local pointer current binding to the address of the current binding pointed to by local pointer path address and sets the local variable postpone to 0. In block **913**, the routine sets the function traverse to the function that retrieves the next data in a list and sets the local pointer session to null. The routine then returns.

FIG. **10** is a flow diagram of the init end routine. If the path is simplex, then the routine creates a new path from where the other one ended, else the routine creates a copy of the path. In block **1001**, if the binding of the passed path entry is simplex (i.e., only one path can be bound to this binding), then the routine continues at block **1002**, else the routine continues at block **1003**. In block **1002**, the routine sets the local pointer path address to point to an address structure that is a copy of the address structure associated with the passed path entry structure with its current binding to the address entry associated with the passed path entry structure, and then returns. In block **1003**, the routine sets the local pointer path address to point to an address structure that contains the URL of the path that contains the passed path entry. In block **1004**, the routine sets the local pointer element to null to initialize the selection of the bindings. In blocks **1005** through **1007**, the routine loops adding all the bindings for the address of the passed path entry that include and are before the passed path entry to the address pointed to by the local path address. In block **1005**, the routine retrieves the next binding from the binding list starting with the first. If there is no such binding, then the routine returns, else the routine continues at block **1006**. In block **1006**, the routine adds the binding to the binding list of the local path address structure and sets the current binding of the local variable path address. In the block **1007**, if the local pointer element is equal to the address entry of the passed path entry, then the routine returns, else the routine loops to block **1005** to select the next binding.

FIG. **11** is a flow diagram of a routine to get the next binding. This routine returns the next binding from the local binding list. If there is no next binding, then the routine invokes the routine label map get to identify the list of edges ("trails") that will map the output label to the target label. If only one trail is identified, then the binding list of path address is extended by the edges of the trail. If multiple trails are identified, then a path is created for each trail and the routine returns so that the demux process can be invoked for each created path. In block **1101**, the routine sets the local pointer binding to point to the next or previous (as indicated by the traverse function) binding in the local binding list. In block **1102**, if a binding was found, then the routine returns an indication that a binding was found, else the routine continues at block **1103**. In block **1103**, the routine invokes the label map get function passing the output label and target label of the local map structure. The label map get function returns a trail list. A trail is a list of edges from the output label to the target label. In decision block **1104**, if the size of the trail list is one, then the routine continues at block **1105**, else the routine continues at block **1112**. In blocks **1105-1111**, the routine extends the binding list by adding a binding data structure for each edge in the trail. The routine then sets the local binding to the last binding in the binding list. In block **1108**, the routine sets the local pointer current binding to point to the last binding in the local binding list. In block

1106, the routine sets the local variable temp trail to the trail in the trail list. In block **1107**, the routine extends the binding list by temp trail by adding a binding for each edge in the trail. These bindings are not yet nailed. In block **1108**, the routine sets the local binding to point to the last binding in the local binding list. In decision block **1109**, if the local binding does not have a key for a session and the local map has a target key for a session, then the routine sets the key for the binding to the target key of the local map and continues at block **1110**, else the routine loops to block **1101** to retrieve the next binding in path. In block **1110**, the routine sets the key of the local binding to the target key of the local map. In block **1111**, the routine sets the target key of the local map to null and then loop to block **1101** to return the next binding. In decision block **1112**, if the local session is set, then the demultiplexing is already in progress and the routine returns a break status. In block **1113**, the routine invokes a prepare multicast paths routine to prepare a path entry for each trail in the trail list. The routine then returns a multiple status.

FIG. **12** is a flow diagram of the get key routine. The get key routine invokes an edge's demuxkey routine to retrieve a key for the session associated with the message. The key identifies the session of a protocol. The demux key routine creates the appropriate key for the message. The demux key routine returns a status of remove, postpone, or other. The status of remove indicates that the current binding should be removed from the path. The status of postpone indicates that the demux key routine cannot create the key because it needs information provided by subsequent protocols in the path. For example, a TCP session is defined by a combination of a remote and local port address and an IP address. Thus, the TCP protocol postpones the creating of a key until the IP protocol identifies the IP address. The get key routine returns a next binding status to continue at the next binding in the path. Otherwise, the routine returns a continue status. In block **1201**, the routine sets the local edge to the edge of the local binding (current binding) and sets the local protocol to the protocol of the local edge. In block **1202**, the routine invokes the demux key routine of the local edge passing the local message, local path address, and local map. The demux key routine sets the key in the local binding. In decision block **1203**, if the demux key routine returns a status of remove, then the routine continues at block **1204**. If the demux key routine returns a status of postpone, then the routine continues at block **1205**, else the routine continues at block **1206**. In block **1204**, the routine sets the flag of the local binding to indicate that the binding is to be removed and continues at block **1206**. In block **1205**, the routine sets the variable traverse to the function to list the next data, increments the variable postpone, and then returns a next binding status. In blocks **1206-1214**, the routine processes the postponing of the creating of a key. In blocks **1207-1210**, if the creating of a key has been postponed, then the routine indicates to backtrack on the path, save the demux status, and set the demux status to demux continue. In blocks **1211-1213**, if the creating of a key has not been postponed, then the routine indicates to continue forward in the path and to restore any saved demux status. The save demux status is the status associated by the binding where the backtrack started. In decision block **1206**, if the variable postpone is set, then the routine continues at block **1207**, else the routine continues at block **1211**. In block **1207**, the routine decrements the variable postpone and sets the variable traverse to the list previous data function. In decision block **1208**, if the variable saved status is set, then the routine continues at block **1210**, else the routine continues at block **1209**. The variable saved status contains the status of the demux process when the demux process started to backtrack.

In block **1209**, the routine sets the variable saved status to the variable status. In block **1210**, the routine sets the variable status to demux continue and continues at block **1214**. In block **1211**, the routine sets the variable traverse to the list next data function. In decision block **1212**, if the variable saved status in set, then the routine continues at block **1213**, else the routine continues at block **1214**. In block **1213**, the routine sets the variable status to the variable saved status and sets the variable saved status to 0. In decision block **1214**, if the local binding indicates that it is to be removed, then the routine returns a next binding status, else the routine returns a continue status.

FIG. **13** is a flow diagram of the get session routine. This routine retrieves the session data structure, creating a data structure session if necessary, for the key indicated by the binding. In block **1301**, the routine retrieves the session from the session table of the local protocol indicated by the key of the local binding. Each protocol maintains a mapping from each key to the session associated with the key. In decision block **1302**, if there is no session, then the routine continues at block **1303**, else the routine returns. In block **1303**, the routine creates a session for the local protocol. In block **1304**, the routine initializes the key for the local session based on the key of the local binding. In block **1305**, the routine puts the session into the session table of the local protocol. In block **1306**, the routine invokes the create session function of the protocol to allow the protocol to initialize its context and then returns.

FIG. **14** is a flow diagram of the nail binding routine. This routine determines whether a binding is already associated with ("nailed to") the session. If so, the routine returns that binding. If not, the routine associates the binding with the session. The routine returns a status of simplex to indicate that only one path can extend through the nailed binding. In decision block **1401**, if the binding table of the session contains an entry for the edge, then the routine continues at block **1402**, else the routine continues at block **1405**. In block **1402**, the routine sets the binding to the entry from the binding table of the local session for the edge. In block **1403**, the routine sets the current binding to point to the binding from the session. In block **1404**, if the binding is simplex, then the routine returns a simplex status, else the routine returns. Blocks **1405** through **1410** are performed when there is no binding in the session for the edge. In block **1405**, the routine sets the session of the binding to the variable session. In block **1406**, the routine sets the key of the binding to the key from the session. In block **1407**, the routine sets the entry for the edge in the binding table of the local session to the binding. In block **1408**, the routine invokes the create binding function of the edge of the binding passing the binding so the edge can initialize the binding. If that function returns a status of remove, the routine continues at block **1409**. In block **1409**, the routine sets the binding to be removed and then returns.

FIG. **15** is a flow diagram of the find path routine. The find path routine identifies the shortest path through the binding list. If no such path exists, then the routine extends a path to include the binding list. In decision block **1501**, if the binding is simplex and a path already goes through this binding (returned as an entry), then the routine continues at block **1502**, else the routine continues at block **1503**. In block **1502**, the routine sets the path to the path of the entry and returns. In block **1503**, the routine initializes the pointers element and short entry to null. In block **1504**, the routine sets the path to the path of the passed path entry. If the local path is not null and its status is demux extend, then the routine continues at block **1509**, else the routine continues at block **1505**. In blocks **1505-1508**, the routine loops identifying the shortest path through the bindings in the binding list. The routine loops selecting each path through the binding. The selected path is eligible if it starts at the first binding in the binding list and the path ends at the binding. The routine loops setting the short entry to the shortest eligible path found so far. In block **1505**, the routine sets the variable first binding to the first binding in the binding list of the path address. In block **1506**, the routine selects the next path (entry) in the path list of the binding starting with the first. If a path is selected (indicating that there are more paths in the binding), then the routine continues at block **1507**, else the routine continues at block **1509**. In block **1507**, the routine determines whether the selected path starts at the first binding in the binding list, whether the selected path ends at the last binding in the binding list, and whether the number of path entries in the selected path is less than the number of path entries in the shortest path selected so far. If these conditions are all satisfied, then the routine continues at block **1508**, else the routine loops to block **1506** to select the next path (entry). In block **1508**, the routine sets the shortest path (short entry) to the selected path and loops to block **1506** to select the next path through the binding. In block **1509**, the routine sets the selected path (entry) to the shortest path. In decision block **1510**, if a path has been found, then the routine continues at block **1511**, else the routine continues at block **1512**. In block **1511**, the routine sets the path to the path of the selected path entry and returns. Blocks **1512-1516** are performed when no paths have been found. In block **1512**, the routine sets the path to the path of the passed path entry. If the passed path entry has a path and its status is demux extend, then the routine continues at block **1515**, else the routine continues at block **1513**. In block **1513**, the routine creates a path for the path address. In block **1514**, the routine sets the variable element to null and sets the path entry to the first element in the stack list of the path. In block **1515**, the routine sets the variable element to be address entry of the member of the passed path entry and sets the path entry to the passed path entry. In block **1516**, the routine invokes the extend path routine to extend the path and then returns. The extend path routine creates a path through the bindings of the binding list and sets the path status to the current demux status.

FIG. **16** is a flow diagram of the process of path hopping routine. Path hopping occurs when the path through the binding list is not the same path as that of the passed path entry. In decision block **1601**, if the path of the passed path entry is set, then the routine continues at block **1602**, else the routine continues at block **1609**. In decision block **1602**, if the path of the passed path entry is equal to the local path, then the routine continues at **1612**, else path hopping is occurring and the routine continues at block **1603**. In blocks **1603-1607**, the routine loops positioning pointers at the first path entries of the paths that are not at the same binding. In block **1603**, the routine sets the variable old stack to the stack list of the path of the passed path entry. In block **1604**, the routine sets the variable new stack to the stack list of the local path. In block **1605**, the routine sets the variable old element to the next element in the old stack. In block **1606**, the routine sets the variable element to the next element in the new stack. In decision block **1607**, the routine loops until the path entry that is not in the same binding is located. In decision block **1608**, if the variable old entry is set, then the routine is not at the end of the hopped from path and the routine continues at block **1609**, else routine continues at block **1612**. In block **1609**, the routine sets the variable entry to the previous entry in the hopped-to path. In block **1610**, the routine sets the path of the passed path entry to the local path. In block **1611**, the routine sets the local entry to the first path entry of the stack list of the local path. In block **1612**, the routine inserts an entry into return list and then returns.

Although the conversion system has been described in terms of various embodiments, the invention is not limited to these embodiments. Modification within the spirit of the invention will be apparent to those skilled in the art. For example, a conversion routine may be used for routing a message and may perform no conversion of the message. Also, a reference to a single copy of the message can be passed to each conversion routine or demuxkey routine. These routines can advance the reference past the header information for the protocol so that the reference is positioned at the next header. After the demux process, the reference can be reset to point to the first header for processing by the conversion routines in sequence. The scope of the invention is defined by the claims that follow.

What is claimed is:

1. A first apparatus for receiving data from a second apparatus, the first apparatus comprising:
- a processing unit; and
- a memory storing instructions executable by the processing unit to:
  - create, based on an identification of information in a received packet of a message, a path that includes one or more data structures that indicate a sequence of routines for processing packets in the message;
  - store the created path; and
  - process subsequent packets in the message using the sequence of routines indicated in the stored path, wherein the sequence includes a routine that is used to execute a Transmission Control Protocol (TCP) to convert one or more packets having a TCP format into a different format.

2. The first apparatus of claim **1**, wherein the sequence includes:
- a second routine that is used to execute a second, different protocol to convert packets of the different format into another format; and
- a third routine that is used to execute a third, different protocol to further convert the packets.

3. The first apparatus of claim **2**, wherein the second protocol is an Internet Protocol (IP) and the third protocol is an Ethernet Protocol.

4. The first apparatus of claim **1**, wherein the one or more data structures further indicate sessions corresponding to respective ones of the sequence of routines.

5. The first apparatus of claim **4**, wherein the sessions specify state information for one or more of the sequence of routines, and wherein the state information is specific to the message.

6. The first apparatus of claim **1**, wherein the sequence of routines includes a routine that is executable to process the packets without converting a format of the packets.

7. The first apparatus of claim **1**, wherein the routine is not executable to convert packets having the different format, and wherein the different format is an Internet Protocol (IP) format.

8. The first apparatus of claim **1**, wherein the memory stores instructions executable by the processing unit to identify an address associated with the information, wherein the address indicates the routines in the sequence of routines of the created path.

9. The first apparatus of claim **8**, wherein the memory stores instructions executable by the processing unit to use the address to select the sequence of routines from a plurality of sequences of routines that are stored by the first apparatus prior to receiving the packet of the message.

10. A non-transitory, computer-readable medium comprising software instructions for processing a message, wherein the software instructions, when executed, cause a computer system to:

obtain information from a particular packet of the message, wherein the particular packet has been received by the computer system;

use the obtained information to identify an address specifying a list of conversion routines;

create a path that includes one or more data structures that specify a sequence of sessions, wherein sessions in the sequence correspond to respective ones of the conversion routines in the list;

store the created path; and

process subsequent packets of the message using sessions specified in the created path, including:

a session associated with a transport layer protocol that is executed to convert one or more packets in a transport layer format into a different format; and

another session associated with a different protocol that is executed, wherein the different protocol corresponds to the different format.

11. The medium of claim **10**, wherein one or more of the sessions specify state information for one or more of the conversion routines, and wherein the state information is specific to the message.

12. The medium of claim **11**, wherein the different protocol is associated with a layer selected from the group consisting of an application layer and a network layer.

13. The medium of claim **10**, wherein the transport layer protocol is a Transmission Control Protocol (TCP).

14. The medium of claim **13**, wherein the message comprises a stream of data.

15. The medium of claim **10**, wherein using the obtained information to identify the address includes determining a plurality of protocols by analyzing headers of the particular packet, and wherein the medium includes software instructions executable to determine protocols executable at the transport layer and at an application layer.

16. A first apparatus configured to receive data from a second apparatus, the first apparatus comprising:

a processing unit; and

memory storing instructions that are executable by the processing unit to:

obtain and analyze information from a received packet of a message;

identify an address based on the obtained information, wherein the address references a list of routines;

create one or more data structures that indicate state information corresponding to routines in the list;

store the one or more data structures; and

process subsequent packets of the message using the state information, including state information that corresponds to a particular routine that is used to execute a protocol to convert packets from an input format to an output format, wherein the particular routine is not executable to convert packets having the output format.

17. The first apparatus of claim **16**, wherein the state information used to process subsequent packets of the message includes state information that corresponds to a different

routine that is used to execute a second, different protocol to convert packets from the output format to a different output format, and wherein the state information used to process subsequent packets of the message includes state information that corresponds to another routine that is used to execute a third, different protocol associated with the different output format.

18. The first apparatus of claim **17**, wherein the protocols include a Transmission Control Protocol (TCP), an Internet Protocol (IP), and an Ethernet Protocol.

19. The first apparatus of claim **16**, wherein at least one of the routines in the list is executable to process packets of the message without converting a format of the packets.

20. The first apparatus of claim **16**, wherein the particular routine is executable to convert packets by removing an outermost header of the packets.

21. The first apparatus of claim **16**, wherein the protocol is a transport layer protocol.

22. The first apparatus of claim **21**, wherein the transport layer protocol is a Transmission Control Protocol (TCP), and wherein the message comprises a stream of data.

23. The first apparatus of claim **16**, wherein the obtained information includes information from headers of the received packet that are associated with a network layer and a transport layer.

24. A non-transitory, computer-readable medium comprising program instructions executable by a computer system to:

identify information from different headers associated with various layers of a received packet of a message;

create, using the identified information, one or more data structures that reference a sequence of routines;

store the one or more data structures; and

process subsequent packets of the message using the sequence of routines referenced by the one or more data structures, including by removing an outermost header of a given packet using a first routine corresponding to a protocol in a first layer and by removing the resulting outermost header using a second routine corresponding to a different protocol in a different layer.

25. The medium of claim **24**, wherein the protocol in the first layer is a Transmission Control Protocol (TCP), and the message comprises a stream of data.

26. The medium of claim **24**, wherein the protocol in the first layer is a transport layer protocol and the different protocol in the different layer is an application layer protocol.

27. The medium of claim **24**, wherein processing subsequent packets of the message further includes using a third routine corresponding to another protocol in another layer to remove the outermost header resulting from use of the second routine, and wherein the layers include a network layer, a transport layer, and an application layer.

28. The medium of claim **24**, wherein at least one of the sequence of routines is not used to remove a header of the packets.

29. The medium of claim **24**, wherein the outermost header has a format that is incompatible with a format of the resulting outermost header, and wherein the outermost header is associated with a network layer protocol.

30. The medium of claim **24**, wherein the one or more data structures further reference state information for one or more of the routines in the sequence of routines.

* * * * *

EXHIBIT 5

US009270790B2

(12) **United States Patent**
Balassanian

(10) **Patent No.: US 9,270,790 B2**
(45) **Date of Patent: Feb. 23, 2016**

(54) **METHOD AND SYSTEM FOR DATA DEMULTIPLEXING**

(71) Applicant: **IMPLICIT, LLC**, Seattle, WA (US)

(72) Inventor: **Edward Balassanian**, Seattle, WA (US)

(73) Assignee: **Implicit, LLC**, Seattle, WA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 49 days.

(21) Appl. No.: **14/230,952**

(22) Filed: **Mar. 31, 2014**

(65) **Prior Publication Data**

US 2015/0009997 A1 Jan. 8, 2015

Related U.S. Application Data

(63) Continuation of application No. 13/911,324, filed on Jun. 6, 2013, now Pat. No. 8,694,683, which is a continuation of application No. 13/236,090, filed on Sep. 19, 2011, now abandoned, which is a continuation of application No. 10/636,314, filed on Aug. 6, 2003, now Pat. No. 8,055,786, which is a continuation of application No. 09/474,664, filed on Dec. 29, 1999, now Pat. No. 6,629,163.

(51) **Int. Cl.**

H04L 29/06	(2006.01)
H04L 12/701	(2013.01)
H04L 29/08	(2006.01)

(52) **U.S. Cl.**

CPC ***H04L 69/08*** (2013.01); ***H04L 29/06*** (2013.01); ***H04L 45/00*** (2013.01); ***H04L 69/22*** (2013.01); ***H04L 69/32*** (2013.01)

(58) **Field of Classification Search**

None

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,298,674	A	3/1994	Yun
5,414,833	A	5/1995	Hershey et al.
5,627,997	A	5/1997	Pearson et al.
5,761,651	A	6/1998	Hasebe
5,826,027	A	10/1998	Pedersen et al.
5,835,726	A	11/1998	Shwed et al.
5,848,233	A	12/1998	Radia et al.
5,848,415	A	12/1998	Guck
5,854,899	A	12/1998	Callon et al.
5,898,830	A	4/1999	Wesinger, Jr. et al.

(Continued)

FOREIGN PATENT DOCUMENTS

EP	0807347	11/1997
EP	0817031	1/1998

OTHER PUBLICATIONS

Alexander, D. et al., "The SwitchWare Active Network Architecture", Jun. 6, 1998, IEEE.

(Continued)

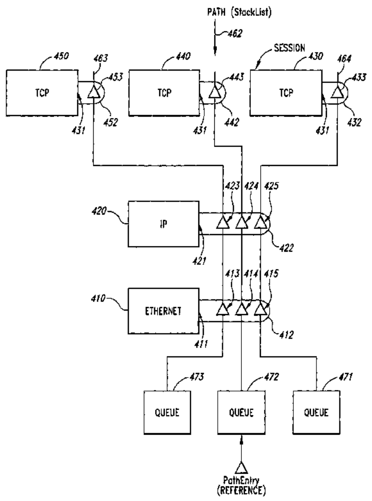
Primary Examiner — Duc Duong

(74) *Attorney, Agent, or Firm* — Meyertons, Hood, Kivlin, Kowert & Goetzel, P.C.

(57) **ABSTRACT**

A method and system for demultiplexing packets of a message is provided. The demultiplexing system receives packets of a message, identifies a sequence of message handlers for processing the message, identifies state information associated with the message for each message handler, and invokes the message handlers passing the message and the associated state information. The system identifies the message handlers based on the initial data type of the message and a target data type. The identified message handlers effect the conversion of the data to the target data type through various intermediate data types.

20 Claims, 16 Drawing Sheets



(56) **References Cited**

U.S. PATENT DOCUMENTS

6,091,725	A	7/2000	Cheriton et al.	
6,104,500	A	8/2000	Alam et al.	
6,115,393	A	9/2000	Engel et al.	
6,119,236	A	9/2000	Shipley	
6,141,749	A	10/2000	Coss et al.	
6,151,390	A	11/2000	Volftsun et al.	
6,226,267	B1	5/2001	Spinney et al.	
6,243,667	B1	6/2001	Kerr et al.	
6,259,781	B1	7/2001	Crouch et al.	
6,356,529	B1	3/2002	Zarom	
6,401,132	B1	6/2002	Bellwood et al.	
6,426,943	B1	7/2002	Spinney et al.	
6,519,636	B2	2/2003	Engel et al.	
6,598,034	B1	7/2003	Kloth	
6,629,163	B1 *	9/2003	Balassanian	H04L 29/06 370/401
6,651,099	B1	11/2003	Dietz et al.	
6,678,518	B2	1/2004	Eerola	
6,680,922	B1	1/2004	Jorgensen	
6,701,432	B1	3/2004	Deng et al.	
6,711,166	B1	3/2004	Amir et al.	
6,785,730	B1	8/2004	Taylor	
6,871,179	B1	3/2005	Kist et al.	
6,889,181	B2	5/2005	Kerr et al.	
7,233,569	B1 *	6/2007	Swallow	H04L 12/4633 370/225
7,281,036	B1 *	10/2007	Lu	H04L 29/12028 709/220
7,383,341	B1	6/2008	Saito et al.	
8,055,786	B2 *	11/2011	Balassanian	H04L 29/06 370/351
8,694,683	B2 *	4/2014	Balassanian	H04L 29/06 370/466
2008/0250045	A1 *	10/2008	Balassanian	G06F 17/30569
2009/0265695	A1 *	10/2009	Karino	G06F 11/3612 717/131

OTHER PUBLICATIONS

Antoniazzi, S. et al., "An Open Software Architecture for Multimedia Consumer Terminals", Central Research Labs, Italy; Alcatel SEL Research Centre, Germany, ECMAST 1997.
Arbanowski, Stefan, "Generic Description of Telecommunication Services and Dynamic Resource Selection in Intelligent Communication Environments", Thesis, Technische Universitat Berlin, Oct. 9, 1996. (3 documents).
Arbanowski, S., et al., Service Personalization for Unified Messaging Systems, Jul. 6-8, 1999, The Fourth IEEE Symposium on Computers and Communications, ISCC '99, Red Sea, Egypt.
Atkinson, R., "Security Architecture for the Internet Protocol", Aug. 1995, Naval Research Laboratory.
Atkinson, R., "IP Authentication Header", Aug. 1995, Naval Research Laboratory.
Atkinson, R., "IP Encapsulating Security Payload (ESP)", Aug. 1995, Naval Research Laboratory.
Back, G., et al., Java Operating Systems: Design and Implementation, Aug. 1998, Technical Report UUCS-98-015, University of Utah.
Baker, Dr. Sean, "CORBA Implementation Issues", 1994, IONA Technologies, O'Reilly Institute Dublin, Ireland.
Barrett, R., et al., "Intermediaries: New Places for Producing and Manipulating Web Content", 1998, IBM Almaden Research Center, Elsevier Science.
Bellare, M., et al., "A Concrete Security Treatment of Symmetric Encryption: Analysis of the DES Modes of Operation", Aug. 15, 1997, Dept. of Computer Science and Engineering, University of California, San Diego.
Bellare, M., et al., "A Concrete Security Treatment of Symmetric Encryption: Analysis of the DES Modes of Operation", Aug. 15, 1997, IEEE.
Bellare, M., et al., "XOR MACs: New Methods for Message Authentication Using Finite Pseudorandom Functions", 1995, CRYPTO '95, LNCS 963, pp. 15-28, Springer-Verlag Berlin Heidelberg.
Bellissard, L., et al., "Dynamic Reconfiguration of Agent-Based Applications", Third European Research Seminar on Advances in Distributed Systems, (ERSADS '99) Madeira Island.
Bolding, Darren, "Network Security, Filters and Firewalls", 1995, www.acm.org/crossroads/xrds2-1/security.html.
Booch, G., et al., "Software Engineering with ADA", 1994, Third Edition, The Benjamin/Cummings Publishing Company, Inc. (2 documents).
Breugst, et al., "Mobile Agents—Enabling Technology for Active Intelligent Network Implementation", May/Jun. 1998, IEEE Network.
"C Library Functions", AUTH(3) Sep. 17, 1993, Solbourne Computer, Inc.
Chapman, D., et al., "Building Internet Firewalls", Sep. 1995, O'Reilly & Associates, Inc.
CheckPoint FireWall-1 Technical White Paper, Jul. 18, 1994, CheckPoint Software Technologies, Ltd.
CheckPoint FireWall-1 White Paper, Sep. 1995, Version 2.0, CheckPoint Software Technologies, Ltd.
Command Line Interface Guide P/N 093-0011-000 Rev C Version 2.5, 2000-2001, NetScreen Technologies, Inc.
Coulson, G. et al., "A CORBA Compliant Real-Time Multimedia Platform for Broadband Networks", Lecture Notes in Computer Science, 1996, Trends in Distributed Systems CORBA and Beyond.
Cox, Brad, "SuperDistribution, Objects As Property on the Electronic Frontier", 1996, Addison-Wesley Publishing Company.
Cranes, et al., "A Configurable Protocol Architecture for CORBA Environments", Autonomous Decentralized Systems 1997 Proceedings ISADS, Third International Symposium Apr. 9-11, 1997.
Curran, K., et al., "CORBA Lacks Venom", University of Ulster, Northern Ireland, UK 2000.
Dannert, Andreas, "Call Logic Service for a Personal Communication Supporting System", Thesis, Jan. 20, 1998, Technische Universitat Berlin.
DARPA Internet Program Protocol Specification, "Transmission Control Protocol", Sep. 1981, Information Sciences Institute, California.
DARPA Internet Program Protocol Specification, "Internet Protocol", Sep. 1981, Information Sciences Institute, California.
Decasper, D., et al., "Crossbow: A Toolkit for Integrated Services over Cell Switched IPv6", 1997, Computer Engineering and Networks Laboratory, ETH Zurich, Switzerland.
Decasper, D., et al., "Router Plugins A Software Architecture for Next Generation Routers", 1998, Proceedings of ACM SIGCONM '98.
Deering, S., et al., Internet Protocol, Version 6 (IPv6) Specification, Dec. 1998, Nokia, The Internet Society.
Deering, S., et al., Internet Protocol, Version 6 (IPv6) Specification, Dec. 1995, Network Working Group, RFC 1883.
Dutton, et al, "Asynchronous Transfer Mode Technical Overview (ATM)", Second Edition; IBM, Oct. 1995, 2nd Edition, Prentice Hall PTR, USA.
Eckardt, T., et al., "Application of X.500 and X.700 Standards for Supporting Personal Communications in Distributed Computing Environments", 1995, IEEE.
Eckardt, T., et al., "Personal Communications Support based on TMN and TINA Concepts", 1996, IEEE Intelligent Network Workshop (IN '96), Apr. 21-24, Melbourne, Australia.
Eckardt, T., et al., "Beyond In and UPT—A Personal Communications Support System Based on TMN Concepts", Sep. 1997, IEEE Journal on Selected Areas in Communications, vol. 15, No. 7.
Egevang, K., et al., "The IP Network Address Translator (NAT)", May 1994, Network Working Group, RFC 1631.
Estrin, D., et al., "Visa Protocols for Controlling Inter-Organizational Datagram Flow", Dec. 1998, Computer Science Department, University of Southern California and Digital Equipment Corporation.
Faupel, M., "Java Distribution and Deployment", Oct. 9, 1997, APM Ltd., United Kingdom.
Felber, P., "The CORBA Object Group Service: A Service Approach to Object Groups in CORBA", Thesis, 1998, Ecole Polytechnique Federale de Lausanne, Switzerland.

(56) **References Cited**

OTHER PUBLICATIONS

Fish, R., et al., "DRoPS: Kernel Support for Runtime Adaptable Protocols", Aug. 25-27, 1998, IEEE 24th Euromicro Conference, Sweden.
Fiuczynski, M., et al., "An Extensible Protocol Architecture for Application-Specific Networking", 1996, Department of Computer Science and Engineering, University of Washington.
Franz, Stefan, "Job and Stream Control in Heterogeneous Hardware and Software Architectures", Apr. 1998, Technische Universitat, Berlin (2 documents).
Fraser, T., "DTE Firewalls: Phase Two Measurement and Evaluation Report", Jul. 22, 1997, Trusted Information Systems, USA.
Gazis, V., et al., "A Survey of Dynamically Adaptable Protocol Stacks", first Quarter 2010, IEEE Communications Surveys & Tutorials, vol. 12, No. 1, 1st Quarter.
Gokhale, A., et al., "Evaluating the Performance of Demultiplexing Strategies for Real-Time CORBA", Nov. 1997, GLOBECOM.
Gokhale, A., et al., "Measuring and Optimizing CORBA Latency and Scalability Over High-Speed Networks", Apr. 1998, IEEE Transaction on Computers, vol. 47, No. 4; Proceedings of the International Conference on Distributed Computing Systems (ICDCS '97) May 27-30, 1997.
Gokhale, A., et al., "Operating System Support for High-Performance, Real-Time CORBA", 1996.
Gokhale, A., et al., "Principles for Optimizing CORBA Internet Inter-ORB Protocol Performance", Jan. 9, 1998, Proceedings of the HICSS Conference, Hawaii.
Gong, Li, "Java Security: Present and Near Future", May/Jun. 1997, IEEE Micro.
Gong, Li, "New Security Architectural Directions for Java (Extended Abstract)", Dec. 19, 1996, IEEE.
Gong, Li, "Secure Java Class Loading", Nov./Dec. 1998, IEEE Internet.
Goos, G., et al., "Lecture Notes in Computer Science: Mobile Agents and Security", 1998, Springer-Verlag Berlin Heidelberg.
Goralski, W., "Introduction to ATM Networking", 1995, McGraw-Hill Series on Computer Communications, USA.
Hamzeh, K., et al., Layer Two Tunneling Protocol "L2TP", Jan. 1998, PPP Working Group, Internet Draft.
Harrison, T., et al., "The Design and Performance of a Real-Time CORBA Event Service", Aug. 8, 1997,Proceedings of the OOPSLA '97 Conference, Atlanta, Georgia in Oct. 1997.
Huitema, Christian, "IPv6 The New Internet Protocol", 1997 Prentice Hall, Second Edition.
Hutchins, J., et al., "Enhanced Internet Firewall Design Using Stateful Filters Final Report", Aug. 1997, Sandia Report; Sandia National Laboratories.
IBM, Local Area Network Concepts and Products: Routers and Gateways, May 1996.
Juniper Networks Press Release, Juniper Networks Announces Junos, First Routing Operating System for High-Growth Internet Backbone Networks, Jul. 1, 1998, Juniper Networks.
Juniper Networks Press Release, Juniper Networks Ships the Industry's First Internet Backbone Router Delivering Unrivaled Scalability, Control and Performance, Sep. 16, 1998, Juniper Networks.
Karn, P., et al., "The ESP DES-CBC Transform", Aug. 1995, Network Working Group, RFC 1829.
Kelsey, J. et al., "Authenticating Outputs of Computer Software Using a Cryptographic Coprocessor", Sep. 1996, CARDIS.
Krieger, D., et al., "The Emergence of Distributed Component Platforms", Mar. 1998, IEEE.
Krupczak, B., et al., "Implementing Communication Protocols in Java", Oct. 1998, IEEE Communications Magazine.
Krupczak, B., et al., "Implementing Protocols in Java: The Price of Portability", 1998, IEEE.
Lawson, Stephen, "Cisco NetFlow Switching Speeds Traffic Routing", Jul. 7, 1997, Infoworld.
Li, S., et al., "Active Gateway: A Facility for Video Conferencing Traffic Control", Feb. 1, 1997, Purdue University; Purdue e-Pubs; Computer Science Technical Reports.
Magedanz, T., et al., "Intelligent Agents: An Emerging Technology for Next Generation Telecommunications?", 1996, IEEE.
Mills, H., et al., "Principles of Information Systems Analysis and Design", 1986, Academic Press, Inc. (2 documents).
Mosberger, David, "*Scout: A Path-Based Operating System*", Doctoral Dissertation Submitted to the University of Arizona, 1997 (3 documents).
Muhugusa, M., et al., "ComScript : An Environment for the Implementation of Protocol Stacks and their Dynamic Reconfiguration", Dec. 1994.
Nelson, M., et al., The Data Compression Book, 2nd Edition, 1996, M&T Books, A division of MIS Press, Inc.
NetRanger User's Guide, 1996, WheelGroup Corporation.
NetScreen Command Line Reference Guide, 2000, P/N 093-0000-001 Rev A, NetScreen Technologies, Inc., USA.
NetScreen Command Line Reference Guide, 2000, P/N 093-0000-001 NetScreen Technologies, Inc., USA.
NetScreen Concepts and Examples ScreenOS Reference Guide, 1998-2001, Version 2.5 P/N 093-0039-000 Rev. A, NetScreen Technologies, Inc.
NetScreen Products Webpage, wysiwyg://body_bottom.3/http://www...een.com/products/products.html 1998-1999, NetScreen Technologies, Inc.
NetScreen WebUI, Reference Guide, Version 2.5.0 P/N 093-0040-000 Rev. A, 2000-2001, NetScreen Technologies, Inc.
NetStalker Installation and User's Guide, 1996, Version 1.0.2, Haystack Labs, Inc.
Niculescu, Dragos, "Survey of Active Network Research", Jul. 14, 1999, Rutgers University.
Nortel Northern Telecom, "ISDN Primary Rate User-Network Interface Specification", Aug. 1998.
Nygren, Erik, "The Design and Implementation of a High-Performance Active Network Node", Thesis, Feb. 1998, MIT.
Osbourne, E., "Morningstar Technologies SecureConnect Dynamic Firewall Filter User's Guide", Jun. 14, 1995, V. 1.4, Morning Star Technologies, Inc.
Padovano, Michael, "Networking Applications on UNIX System V Release 4," 1993 Prentice Hall, USA (2 documents).
Pfeifer, T., "Automatic Conversion of Communication Media", 2000, GMD Research Series, Germany.
Pfeifer, T., "Automatic Conversion of Communication Media", Thesis, 1999, Technischen Universitat Berlin, Berlin.
Pfeifer, T., et al., "Applying Quality-of-Service Parametrization for Medium-to-Medium Conversion", Aug. 25-28, 1996, 8th IEEE Workshop on Local and Metropolitan Area Networks, Potsdam, Germany.
Pfeifer, T., "Micronet Machines—New Architectural Approaches for Multimedia End-Systems", 1993 Technical University of Berlin.
Pfeifer, T., "On the Convergence of Distributed Computing and Telecommunications in the Field of Personal Communications", 1995, KiVS, Berlin.
Pfeifer, T., "Speech Synthesis in the Intelligent Personal Communication Support System (IPCSS)", Nov. 2-3, 1995, 2nd 'Speak!' Workshop on Speech Generation in Multimodal Information Systems and Practical Applications.
Pfeifer, T., et al., "Generic Conversion of Communication Media for Supporting Personal Mobility", Nov. 25-27, 1996, Proc. of the Third COST 237 Workshop: Multimedia Telecommunications and Applications.
Pfeifer, T., et al., "Intelligent Handling of Communication Media", Oct. 29-31, 1997, 6th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS) Tunis.
Pfeifer, T., et al., "Resource Selection in Heterogeneous Communication Environments using the Teleservice Descriptor", Dec. 15-19, 1997, Proceedings from the 4th COST 237 Workshop: From Multimedia Services to Network Services, Lisboa.
Pfeifer, T., et al., Mobile Guide—Location-Aware Applications from the Lab to the Market, 1998, IDMS '98, LNCS 1483, pp. 15-28.
Pfeifer, T., et al., "The Active Store providing Quality Enhanced Unified Messaging", Oct. 20-22, 1998, 5th Conference on computer Communications, AFRICOM-CCDC '98, Tunis.
Pfeifer, T.,, et al., "*A Modular Location-Aware Service and Application Platform*", 1999, Technical University of Berlin.

(56) **References Cited**

OTHER PUBLICATIONS

Plagemann, T., et al., "*Evaluating Crucial Performance Issues of Protocol Configuration in DaCaPo*", 1994, University of Oslo.
Psounis, Konstantinos, "*Active Networks: Applications, Security, Safety, and Architectures*", First Quarter 1999, IEEE Communications Surveys.
Rabiner, Lawrence, "*Applications of Speech Recognition in the Area of Telecommunications*", 1997, IEEE.
Raman, Suchitra, et al, "*A Model, Analysis, and Protocol Framework for Soft State-based Communications*", Department of EECS, University of California, Berkeley.
Rogaway, Phillip, "*Bucket Hashing and its Application to Fast Message Authentication*", Oct. 13, 1997, Department of Computer Science, University of California.
Schneier, B., et al., "*Remote Auditing of Software Outputs Using a Trusted CoProcessor*", 1997, Elsevier Paper Reprint 1999.
Tennenhouse, D., et al., "*From Internet to ActiveNet*", Laboratory of Computer Science, MIT, 1996.
Tudor, P., "*Tutorial MPEG-2 Video Compression*", Dec. 1995, Electronics & Communication Engineering Journal.
US Copyright Webpage of Copyright Title, "*IPv6: the New Internet Protocol*", by Christian Huitema, 1998 Prentice Hall.
Van der Meer, et al., "*An Approach for a 4th Generation Messaging System*", Mar. 21-23, 1999, The Fourth International Symposium on Autonomous Decentralized Systems ISADS '99, Tokyo.
Van der Meer, Sven, "*Dynamic Configuration Management of the Equipment in Distributed Communication Environments*", Thesis, Oct. 6, 1996, Berlin (3 documents).
Van Renesse, R. et al., "*Building Adaptive Systems Using Ensemble*", Cornell University Jul. 1997.
Venkatesan, R., et al., "*Threat-Adaptive Security Policy*", 1997, IEEE.
Wetherall, D., et al., "*The Active IP Option*", Sep. 1996, Proceedings of the 7th ACM SIGOPS European Workshop, Connemara, Ireland.
Welch, Terry, "*A Technique for High-Performance Data Compression*", 1984, Sperry Research Center, IEEE.
Zeletin, R. et al., "*Applying Location-Aware Computing for Electronic Commerce: Mobile Guide*", Oct. 20-22, 1998, 5th Conference on Computer Communications, AFRICOM-CCDC '98, Tunis.
Zell, Markus, "*Selection of Converter Chains by Means of Quality of Service Analysis*", Thesis, Feb. 12, 1998, Technische Universitat Berlin.
Implicit Networks, Inc. v. *Advanced Micro Devices, Inc. et al.*; C08-0184 JLR; USDC for the Western District of Washington, Seattle Division.
Feb. 4, 2008 Plaintiff's Original Complaint.
Aug. 26, 2008 Defendant NVIDIA Corporation's Answer to Complaint.
Aug. 26, 2008 Defendant Sun Microsystems, Inc.'s Answer to Complaint.
Aug. 27, 2008 Defendant Advanced Micro Devices, Inc.'s Answer to Complaint for Patent Infringement.
Aug. 27, 2008 RealNetworks, Inc.'s Answer to Implicit Networks, Inc.'s Original Complaint for Patent Infringement, Affirmative Defenses, and Counterclaims.
Aug. 27, 2008 Intel Corp.'s Answer, Defenses and Counterclaims.
Aug. 27, 2008 Defendant RMI Corporation's Answer to Plaintiffs Original Complaint.
Sep. 15, 2008 Plaintiffs Reply to NVIDIA Corporation's Counterclaims.
Sep. 15, 2008 Plaintiffs Reply to Sun Microsystems Inc.'s Counterclaims.
Sep. 16, 2008 Plaintiffs Reply to RealNetworks, Inc.'s Counterclaims.
Sep. 16, 2008 Plaintiffs Reply to Intel Corp.'s Counterclaims.
Dec. 10, 2008 Order granting Stipulated Motion for Dismissal with Prejudice re NVIDIA Corporation, Inc.
Dec. 16, 2008 Defendants AMD, RealNetworks, RMI, and Sun's Motion to Stay Pending the Patent and Trademark Office's Reexamination of the '163 Patent.
Dec. 29, 2008 Order granting Stipulated Motion for Dismissal without Prejudice of Claims re Sun Microsystems, Inc.
Jan. 5, 2009 Plaintiff's Opposition to Defendants AMD, RealNetworks, RMI, and Sun's Motion to Stay Pending Reexamination and Exhibit A.
Jan. 9, 2009 Reply of Defendants AMD, RealNetworks, RMI, and Sun's Motion to Stay Pending the Patent and Trademark Office's Reexamination of the '163 Patent.
Feb. 9, 2009 Order Granting Stay Pending the United States Patent and Trademark Office's Reexamination of U.S. Pat. No. 6,629,163.
Feb. 17, 2009 Order Granting Stipulated Motion for Dismissal of Advanced Micro Devices, Inc. with Prejudice.
May 14, 2009 Order Granting Stipulated Motion for Dismissal of RMI Corporation with Prejudice.
Oct. 13, 2009 Order Granting Stipulated Motion for Dismissal of Claims Against and Counterclaims by Intel Corporation.
Oct. 30, 2009 Executed Order for Stipulated Motion for Dismissal of Claims Against and Counterclaims by RealNetworks, Inc.
Implicit Networks, Inc. v. *Microsoft Corp.*, C09-5628 HLR; USDC for the Northern District of California, San Francisco Division.
Nov. 30, 2009 Plaintiffs Original Complaint, *Implicit* v *Microsoft*, Case No. 09-5628.
Jan. 22, 2010 Order Dismissing Case, *Implicit* v *Microsoft*, Case No. 09-5628.
Implicit Networks, Inc. v. *Cisco Systems, Inc.*, C10-3606 HRL; USDC for the Northern District of California, San Francisco Division.
Aug. 16, 2010 Plaintiffs Original Complaint, *Implicit* v *Cisco*, Case No. 10-3606.
Nov. 22, 2010 Defendant Cisco Systems, Inc.'s Answer and Counterclaims, *Implicit* v *Cisco*, Case No. 10-3606.
Dec. 13, 2010 Plaintiff, Implicit Networks, Inc.'s, Answer to Counterclaims, *Implicit* v *Cisco*, Case No. 10-3606.
Oct. 4, 2011 Order of Dismissal with Prejudice, *Implicit* v *Cisco*, Case No. 10-3606.
Implicit Networks, Inc. v. *Citrix Systems, Inc.*, C10-3766 JL; USDC for the Northern District of California, San Francisco Division.
Aug. 24, 2010 Plaintiffs Original Complaint, *Implicit* v *Citrix*, Case No. 10-3766.
Dec. 1, 2010 Plaintiff's First Amended Complaint, *Implicit* v *Citrix*, Case No. 10-3766.
Jan. 14, 2011 Defendant Citrix Systems, Inc.'s Answer, Defenses and Counter-complaint for Declaratory Judgment, *Implicit* v *Citrix*, Case No. 10-3766.
Feb. 18, 2011 Plaintiff, Implicit Networks, Inc.'s, Answer to Defendants Counterclaims, *Implicit* v *Citrix*, Case No. 10-3766.
May 2, 2011 Order of Dismissal, *Implicit* v *Citrix*, Case No. 10-3766.
Implicit Networks, Inc. v. *F5 Networks, Inc.*, C10-3365 JCS; USDC for the Northern District of California, San Francisco Division.
Jul. 30, 2010 Plaintiffs Original Complaint, *Implicit* v *F5*, Case No. 10-3365.
Oct. 13, 2010 Defendants' Answer and Counter-Complaint, *Implicit* v *F5*, Case No. 10-3365.
Nov. 3, 2010 Plaintiff's Answer to Counter-Complaint, *Implicit* v *F5*, Case No. 10-3365.
Dec. 10, 2010 Plaintiff's First Amended Complaint, *Implicit* v *F5*, Case No. 10-3365.
Jan. 14, 2011 Defendants' Answer to 1st Amended Complaint and Counterclaim, *Implicit* v *F5*, Case No. 10-3365.
Feb. 18, 2011 Plaintiffs Answer to F5's Amended Counter-Complaint, *Implicit* v *F5*, Case No. 10-3365.
Apr. 18, 2011 Defendants' Amended Answer to 1st Amended Complaint and Counter-Complaint, *Implicit* v *F5*, Case No. 10-3365.
May 5, 2011 Plaintiffs Answer to F5's Amended Counter-Complaint, *Implicit* v *F5*, Case No. 10-3365.
Jul. 22, 2011 F5 Networks, Inc.'s Invalidity Contentions, *Implicit* v *F5*, Case No. 10-3365.
Jul. 22, 2011 F5 Networks, Inc.'s Invalidity Contentions, Exhibit A, *Implicit* v *F5*, Case No. 10-3365 (31 documents).
Jul. 22, 2011 F5 Networks, Inc.'s Invalidity Contentions, Exhibit B, *Implicit* v *F5*, Case No. 10-3365.
Oct. 18, 2011 Joint Claim Construction & Pre-Hearing Statement (PR 4-3), *Implicit* v *F5*, Case No. 10-3365.

(56) **References Cited**

OTHER PUBLICATIONS

Oct. 18, 2011 Joint Claim Construction & Pre-Hearing Statement (PR 4-3) Exhibit A, *Implicit* v *F5*, Case No. 10-3365 (2 documents).
Nov. 28, 2011 Plaintiffs Opening Claim Construction Brief, *Implicit* v *F5*, Case No. 10-3365.
Nov. 29, 2011 Amended Joint Claim Construction & Pre-Hearing Statement, *Implicit* v *F5*, Case No. 10-3365.
Nov. 29, 2011 Amended Joint Claim Construction & Pre-Hearing Statement, Exhibit A, *Implicit* v *F5*, Case No. 10-3365.
Dec. 12, 2011 Defendants' Claim Construction Brief, *Implicit* v *F5*, Case No. 10-3365.
Dec. 19, 2011 Plaintiffs Reply to Defendants' (F5, HP, Juniper) Responsive Claim Construction Brief (4-5), *Implicit* v *F5*, Case No. 10-3365.
Jan. 27, 2012 Transcript of Proceeding Held on Jan. 17, 2012; *Implicit* v *F5*, Case No. 10-3365.
Jan. 27, 2012 Transcript of Proceeding Held on Jan. 18, 2012; *Implicit* v *F5*, Case No. 10-3365.
Jan. 27, 2012 Transcript of Proceeding Held on Jan. 19, 2012; *Implicit* v *F5*, Case No. 10-3365.
Feb. 29, 2012 Claim Construction Order.
Aug. 15, 2012 Storer Invalidity Report.
Sep. 10, 2012 Implicit's Expert Report of Scott M. Nettles.
Mar. 13, 2013 Order Granting Defendants' Motion for Summary Judgment.
Apr. 9, 2013 Notice of Appeal to the Federal Circuit.
Implicit Networks, Inc. v. *Hewlett-Packard Company*, C10-3746 JCS: USDC for the Northern District of California, San Francisco Division.
Aug. 23, 2010 Plaintiffs Original Complaint, *Implicit* v *HP*, Case No. 10-3746.
Nov. 23, 2010 Plaintiff's First Amended Complaint, *Implicit* v *HP*, Case No. 10-3746.
Jan. 14, 2011 Defendant HP's Answer and Counterclaims, *Implicit* v *HP*, Case No. 10-3746.
Feb. 18, 2011 Implicit Networks, Inc.'s Answer to HP Counter-claims, *Implicit* v *HP*, Case No. 10-3746.
May 10, 2011 Plaintiff's Amended Disclosure of Asserted Claims and Infringement Contentions, Case No. 10-3746.
Jun. 30, 2011 Defendant HP Company's Invalidity Contentions, *Implicit* v *HP*, Case No. 10-3746.
Jun. 30, 2011 Defendant HP Company's Invalidity Contentions, A1-14, *Implicit* v *HP*, Case No. 10-3746.
Jun. 30, 2011 Defendant HP Company's Invalidity Contentions, B1-21, *Implicit* v *HP*, Case No. 10-3746.
Implicit Networks, Inc. v. *Juniper Networks*, C10-4234 EDL: USDC for the Northern District of California, San Francisco Division.
Sep. 20, 2010 Plaintiff's Original Complaint, *Implicit* v *Juniper*, Case No. 10-4234.
Nov. 12, 2010 Juniper Network's Motion to Dismiss For Failure to State a Claim Under Rule 12(B)(6): Memorandum of Points and Authorities; *Implicit* v *Juniper*, Case No. 10-4234.
Nov. 12, 2010 Juniper Network's Request for Judicial Notice in Support of its Motion to Dismiss For Failure to State a Claim Under Rule 12(B)(6): Memorandum of Points and Authorities; *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 1, 2010 First Amended Complaint; *Implicit* v *Juniper*, Case No. 10-4234.
Jan. 18, 2011 Juniper Networks, Inc.'s Answer and Affirmative Defenses to 1st Amended Complaint, *Implicit* v *Juniper*, Case No. 10-4234.
Feb. 18, 2011 Plaintiffs Answer to Defendant's Counterclaims, *Implicit* v *Juniper*, Case No. 10-4234.
May 23, 2011 Plaintiffs Disclosure of Asserted Claims and Infringement Contentions, *Implicit* v *Juniper*, Case No. 10-4234.
Nov. 15, 2011 Plaintiffs Amended Disclosure of Asserted Claim and Infringement Contentions, *Implicit* v *Juniper*, Case No. 10-4234.
Nov. 28, 2011 Spencer Hosie Declaration in Support of Plaintiffs Opening Claim Construction Brief), *Implicit* v *Juniper*, Case No. 10-4234.
Nov. 28, 2011 Spencer Hosie Declaration in Support of Plaintiffs Opening Claim Construction Brief Exhibit E, *Implicit* v *Juniper*, Case No. 10-4234.
Nov. 28, 2011 Spencer Hosie Declaration in Support of Plaintiffs Opening Claim Construction Brief Exhibit J, *Implicit* v *Juniper*, Case No. 10-4234.
Nov. 28, 2011 Spencer Hosie Declaration in Support of Plaintiffs Opening Claim Construction Brief Exhibit K, *Implicit* v *Juniper*, Case No. 10-4234.
Nov. 28, 2011 Spencer Hosie Declaration in Support of Plaintiffs Opening Claim Construction Brief Exhibits M-O, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit B, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit F, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit N, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit P, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit Q, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit S., *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit T-1, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit T-2, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit T-3, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit T-4, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit U, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit V, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit W, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit X, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit Y-1, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit Y-2, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit Y-3, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit Y-4, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit Z, *Implicit* v *Juniper*, Case No. 10-4234.

(56) **References Cited**

OTHER PUBLICATIONS

Dec. 19, 2011 Spencer Hosie Declaration in Support of Plaintiff's Reply Claim Construction Brief, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 19, 2011 Spencer Hosie Declaration in Support of Plaintiffs Reply Claim Construction Brief, Exhibit P, *Implicit* v *Juniper*, Case No. 10-4234.
Jan. 10, 2012 Plaintiff's Jan. 10, 2012 Amended Disclosure of Asserted Claims and Infringement Contentions, *Implicit* v *Juniper*, Case No. 10-4234.
Feb. 10, 2012 Juniper Networks, Inc.'s Supplemental Invalidity Contentions, *Implicit* v *Juniper*, Case No. 10-4234.
Feb. 10, 2012 Juniper Networks, Inc.'s Supplemental Invalidity Contentions, Exhibit A1, *Implicit* v *Juniper*, Case No. 10-4234.
Feb. 10, 2012 Juniper Networks, Inc.'s Supplemental Invalidity Contentions, Exhibit A2, *Implicit* v *Juniper*, Case No. 10-4234.
Feb. 10, 2012 Juniper Networks, Inc.'s Supplemental Invalidity Contentions, Exhibit A3, *Implicit* v *Juniper*, Case No. 10-4234.
Feb. 10, 2012 Juniper Networks, Inc.'s Supplemental Invalidity Contentions, Exhibit A4, *Implicit* v *Juniper*, Case No. 10-4234.
Feb. 10, 2012 Juniper Networks, Inc.'s Supplemental Invalidity Contentions, Exhibit B1, *Implicit* v *Juniper*, Case No. 10-4234.
Feb. 29, 2012 Plaintiff's Feb. 29, 2012 Amended Disclosure of Asserted Claims and Infringement Contentions, *Implicit* v *Juniper*, Case No. 10-4234.
Apr. 6, 2012 Plaintiff's Apr. 6, 2012 Amended Disclosure of Asserted Claims and Infringement Contentions, *Implicit* v *Juniper*, Case No. 10-4234.
Apr. 9, 2012 Plaintiff's Apr. 9, 2012 Amended Disclosure of Asserted Claims and Infringement Contentions, *Implicit* v *Juniper*, Case No. 10-4234.
Sep. 11, 2012 Implicit's Expert Report of Scott Nettles.
Nov. 9, 2012 Juniper's Notice of Motion and Memorandum of Law ISO Motion for Summary Judgment or, in the alternative, for Partial Summary Judgment, on the Issue of Invalidity.
Nov. 9, 2012 Exhibit 2 to Declaration in support of Juniper's Motion for Summary Judgment—Calvert Expert Report.
Nov. 9, 2012 Exhibit 3 to Declaration in support of Juniper's Motion for Summary Judgment—Calvert Supplemental Expert Report.
Nov. 26, 2012 Implicit Opposition to Juniper's and F5 Motion on Invalidity.
Nov. 26, 2012 Exhibit A to Hosie Declaration—Aug. 27, 2012 Excerpts from David Blaine deposition.
Nov. 26, 2012 Exhibit B to Hosie Declaration—Oct. 25, 2012 Excerpts from Kenneth Calvert Deposition.
Nov. 26, 2012 Exhibit C to Hosie Declaration—Aug. 15, 2012 Excerpts from Kenneth Calvert Expert Report.
Nov. 26, 2012 Exhibit D to Hosie Declaration—U.S. Pat. No. 6,651,099 to Dietz et al.
Nov. 26, 2012 Exhibit E to Hosie Declaration—Understanding Packet-Based and Flow-Based Forwarding.
Nov. 26, 2012 Exhibit F to Hosie Declaration—Wikipedia on Soft State.
Nov. 26, 2012 Exhibit G to Hosie Declaration—Sprint Notes.
Nov. 26, 2012 Exhibit H to Hosie Declaration—Implicit's Supplemental Response to Juniper's 2nd Set of Interrogatories.
Nov. 26, 2012 Exhibit I to Hosie Declaration—U.S. Pat. No. 7,650,634 (Zuk).
Other Implicit Networks, Inc. Prosecution Matters.
U.S. Appl. No. 11/933,022 Utility Application filed Oct. 31, 2007.
U.S. Appl. No. 11/933,022 Preliminary Amendment filed Feb. 19, 2008.
U.S. Appl. No. 11/933,022 Office Action mailed Jun. 24, 2009.
U.S. Appl. No. 11/933,022 Amendment filed Sep. 24, 2009.
U.S. Appl. No. 11/933,022 Office Action dated Dec. 11, 2009.
U.S. Appl. No. 11/933,022 Amendment and Response dated Jan. 29, 2010.
U.S. Appl. No. 11/933,022 Notice of Allowance dated Mar. 2, 2010.
U.S. Appl. No. 11/933,022 Issue Notification dated May 4, 2010.
U.S. Appl. No. 10/636,314 Utility Application filed Aug. 6, 2003.
U.S. Appl. No. 10/636,314 Office Action dated Apr. 7, 2008.
U.S. Appl. No. 10/636,314 Response to Restriction Requirement dated Aug. 5, 2008.
U.S. Appl. No. 10/636,314 Office Action dated Oct. 3, 2008.
U.S. Appl. No. 10/636,314 Response to Office Action dated Apr. 3, 2009.
U.S. Appl. No. 10/636,314 Notice of Non-Compliant Amendment dated May 4, 2009.
U.S. Appl. No. 10/636,314 Amendment to Office Action Response dated Jun. 4, 2009.
U.S. Appl. No. 10/636,314 Notice of Non-Compliant Amendment dated Jun. 12, 2009.
U.S. Appl. No. 10/636,314 Amendment to Office Action dated Jul. 10, 2009.
U.S. Appl. No. 10/636,314 Final Rejection Office Action dated Oct. 21, 2009.
U.S. Appl. No. 10/636,314 Amendment after Final Office Action dated Dec. 14, 2009.
U.S. Appl. No. 10/636,314 Advisory Action dated Jan. 11, 2010.
U.S. Appl. No. 10/636,314 Notice of Non-Compliant Amendment dated Jan. 11, 2010.
U.S. Appl. No. 10/636,314 Supplemental Amendment and Response dated Mar. 13, 2010.
U.S. Appl. No. 10/636,314 Office Action dated May 11, 2010.
U.S. Appl. No. 10/636,314 Amendment and Response dated Sep. 13, 2010.
U.S. Appl. No. 10/636,314 Final Rejection dated Nov. 24, 2010.
U.S. Appl. No. 10/636,314 Notice of Appeal dated May 19, 2011.
U.S. Appl. No. 10/636,314 Amendment and Request for Continued Examination dated Jul. 19, 2011.
U.S. Appl. No. 10/636,314 Notice of Allowance dated Sep. 13, 2011.
U.S. Appl. No. 10/636,314 Notice of Allowance dated Sep. 19, 2011.
U.S. Appl. No. 10/636,314 Issue Notification dated Oct. 19, 2011.
U.S. Appl. No. 09/474,664 Utility Application filed Dec. 29, 1999.
U.S. Appl. No. 09/474,664 Office Action dated Sep. 23, 2002.
U.S. Appl. No. 09/474,664 Amendment and Response dated Feb. 24, 2003.
U.S. Appl. No. 09/474,664 Notice of Allowance dated May 20, 2003.
U.S. Appl. No. 90/010,356 Request for Ex Parte Reexamination dated Dec. 15, 2008.
U.S. Appl. No. 90/010,356 Office Action Granting Reexamination dated Jan. 17, 2009.
U.S. Appl. No. 90/010,356 First Office Action dated Jul. 7, 2009.
U.S. Appl. No. 90/010,356 First Office Action Response dated Sep. 1, 2009.
U.S. Appl. No. 90/010,356 Patent Owner Interview Summary dated Oct. 23, 2009.
U.S. Appl. No. 90/010,356 Office Action Final dated Dec. 4,2009.
U.S. Appl. No. 90/010,356 Amendment and Response to Office Action dated Dec. 18, 2009.
U.S. Appl. No. 90/010,356 Amendment and Response to Office Action dated Jan. 4, 2010.
U.S. Appl. No. 90/010,356 Advisory Action dated Jan. 21, 2010.
U.S. Appl. No. 90/010,356 Amendment and Response to Advisory Action dated Feb. 8, 2010.
U.S. Appl. No. 90/010,356 Notice of Intent to Issue a Reexam Certificate dated Mar. 2, 2010.
U.S. Appl. No. 90/010,356 Reexamination Certificate Issued dated Jun. 22, 2010.
U.S. Appl. No. 95/000,659 Inter Partes Reexam Request dated Feb. 13, 2012.
U.S. Appl. No. 95/000,659 Order Granting Reexamination dated Apr. 3, 2012.
U.S. Appl. No. 95/000,659 Office Action dated Apr. 3, 2012.
U.S. Appl. No. 95/000,659 Office Action Response dated Jun. 4, 2012 (including Exhibits 1 & 2) (4 documents).
U.S. Appl. No. 95/000,659 Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012.
U.S. Appl. No. 95/000,659 Appendix R-1 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Declaration of Prof. Dr. Bernhard Plattner).

(56) **References Cited**

OTHER PUBLICATIONS

U.S. Appl. No. 95/000,659 Appendix R-2 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Prof. Dr. Bernhard Plattner CV).
U.S. Appl. No. 95/000,659 Appendix R-3 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Listing of Publications to Prof. Dr. Bernhard Plattner updated Feb. 2012).
U.S. Appl. No. 95/000,659 Appendix R-4 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012(Office Action Granting Reexamination in 95/000,660 dated May 10, 2012).
U.S. Appl. No. 95/000,659 Appendix R-5 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Office Action in 95/000,660 dated May 10, 2012).
U.S. Appl. No. 95/000,659 Appendix R-6 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Implicit Networks, Inc. U.S. Pat. No. 6,629,163 Claims Chart).
U.S. Appl. No. 95/000,659 Appendix R-7 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Internet Protocol DARPA Internet Program Protocol Specification dated Sep. 1991).
U.S. Appl. No. 95/000,659 Appendix R-8 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Atkinson, IP Encapsulating Security Payload (ESP) dated Aug. 1995).
U.S. Appl. No. 95/000,659 Appendix R-9 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Claim Construction Order dated Feb. 29, 2012).
U.S. Appl. No. 95/000,659 Appendix R-10-1 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (vol. I of Edward Balassanian Deposition Transcript dated May 30, 2012).
U.S. Appl. No. 95/000,659 Appendix R-10-2 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (vol. II of Edward Balassanian Deposition Transcript dated May 31, 2012).
U.S. Appl. No. 95/000,659 Appendix R-10-3 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (vol. III of Edward Balassanian Deposition Transcript dated Jun. 7, 2012).
U.S. Appl. No. 95/000,659 Appendix R-10-4 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (vol. IV of Edward Balassanian Deposition Transcript dated Jun. 8, 2012).
U.S. Appl. No. 95/000,659 Appendix R-11 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Implicit Networks, Inc.'s Response to Juniper Networks, Inc.'s First Set of Requests for Admission 1-32).
U.S. Appl. No. 95/000,659 Action Closing Prosecution dated Oct. 1, 2012.
U.S. Appl. No. 95/000,659 Petition to Withdraw and Reissue Action Closing Prosecution dated Nov. 20, 2012.
U.S. Appl. No. 95/000,659 Patent Owner Comments to Action Closing Prosecution dated Dec. 3, 2012.
U.S. Appl. No. 95/000,659 Opposition to Petition dated Dec. 17, 2012.
U.S. Appl. No. 95/000,659 Third Party Comments to Action Closing Prosecution dated Jan. 2, 2013.
U.S. Appl. No. 95/000,660 Inter Partes Reexam Request dated Mar. 2, 2012.
U.S. Appl. No. 95/000,660 Order Granting Reexamination dated May 10, 2012.
U.S. Appl. No. 95/000,660 Office Action dated May 10, 2012.
U.S. Appl. No. 95/000,660 Response to Office Action dated Jul. 10, 2012 (including Exhibits 1 and 2).
U.S. Appl. No. 95/000,660 Third Party Comments to Office After Patent Owner's Response dated Aug. 8, 2012 (including Revised Comments).
U.S. Appl. No. 95/000,660 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Declaration of Prof. Dr. Bernhard Plattner).
U.S. Appl. No. 95/000,660 Appendix R-1 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Prof. Dr. Bernhard Plattner CV).
U.S. Appl. No. 95/000,660 Appendix R-3 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Listing of Publications to Prof. Dr. Bernhard Plattner updated Feb. 2012).
U.S. Appl. No. 95/000,660 Appendix R-4 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012(Office Action Granting Reexamination in 95/000,660 dated May 10, 2012).
U.S. Appl. No. 95/000,660 Appendix R-5 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Office Action in 95/000,660 dated May 10, 2012).
U.S. Appl. No. 95/000,660 Appendix R-6 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Implicit Networks, Inc. U.S. Pat. No. 6,629,163 Claims Chart).
U.S. Appl. No. 95/000,660 Appendix R-7 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Internet Protocol DARPA Internet Program Protocol Specification dated Sep. 1991).
U.S. Appl. No. 95/000,660 Appendix R-8 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Atkinson, IP Encapsulating Security Payload (ESP) dated Aug. 1995).
U.S. Appl. No. 95/000,660 Appendix R-9 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Claim Construction Order dated Feb. 29, 2012).
U.S. Appl. No. 95/000,660 Appendix R-10 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (vol. I-IV of Edward Balassanian Deposition Transcript dated May 30, 2012).
U.S. Appl. No. 95/000,660 Appendix R-11 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Shacham, A., et al, "*IP Payload Compression Protocol*", Network Working Group, RFC 3173 Sep. 2001).
U.S. Appl. No. 95/000,660 Appendix R-12 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Shacham, A., et al, "*IP Payload Compression Protocol*", Network Working Group, RFC 2393 Dec. 1998).
U.S. Appl. No. 95/000,660 Appendix R-13 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 ('163 Pfeiffer Claim Chart).
U.S. Appl. No. 95/000,660 Appendix R-14 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Ylonen, T., "*SSH Transport Layer Protocol*", Network Working Group—Draft Feb. 22, 1999).
U.S. Appl. No. 95/000,660 Appendix R-15 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Dommety, G., "*Key and Sequence Number Extensions to GRE*", Network Working Group, RFC 2890 Sep. 2000).
U.S. Appl. No. 95/000,660 Appendix R-16 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Monsour, R., et al, "*Compression in IP Security*" Mar. 1997).
U.S. Appl. No. 95/000,660 Appendix R-17 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Friend, R., Internet Working Group RFC 3943 dated Nov. 2004 *Transport Layer Security Protocol Compression Using Lempel-Ziv-Stac*).
U.S. Appl. No. 95/000,660 Appendix R-18 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012(Implicit Networks, Inc.'s Response to Juniper Networks, Inc.'s First Set of Requests for Admission 1-32).
U.S. Appl. No. 95/000,660 Revised—Third Party Comments to Office After Patent Owner's Response dated Nov. 2, 2012.
U.S. Appl. No. 95/000,660 Action Closing Prosecution dated Dec. 21, 2012.
U.S. Appl. No. 95/000,660 Comments to Action Closing Prosecution dated Feb. 21, 2013 (including Dec of Dr. Ng).

(56) **References Cited**

OTHER PUBLICATIONS

U.S. Appl. No. 95/000,660 Third Party Comments to Action Closing Prosecution dated Mar. 25, 2013.
PCT/US00/33634—PCT application (WO 01/2077 A2—Jul. 12, 2001).
PCT/US00/33634—Written Opinion (WO 01/50277 A3—Feb. 14, 2002).
PCT/US00/33634—International Search Report (Oct. 9, 2001).
PCT/US00/33634—Response to Official Communication dated Dec. 7, 2001 (Mar. 21, 2002).
PCT/US00/33634—International Preliminary Examination Report (Apr. 8, 2002).
PCT/US00/33634—Official Communication (Jan. 24, 2003).
PCT/US00/33634—Response to Official Communication dated Jan. 24, 2003 (Mar. 12, 2003).
PCT/US00/33634—Official Communication (May 13, 2004).
PCT/US00/33634—Response to Summons to Attend Oral Proceeding dated May 13, 2004 (Oct. 9, 2004).
PCT/US00/33634—Decision to Refuse a European Patent application (Nov. 12, 2004).
PCT/US00/33634—Minutes of the oral proceedings before the Examining Division (Oct. 12, 2004).
PCT/US00/33634—Closure of the procedure in respect to Application No. 00984234.5-2212 (Feb. 22, 2005).
May 3, 2013 Expert Report of Dr. Alfonso Cardenas Regarding Validity of U.S. Pat. Nos. 6,877,006; 7,167,864; 7,720,861; and 8,082,268 (6 documents).
Expert Report of Dr. Alfonso Cardenas Regarding Validity of U.S. Pat. No. 7,167,864 (3 documents).
"InfoReports User Guide: Version 3.3.1;" Platinum Technology, Publication No. PRO-X-331-UG00-00, printed Apr. 1998; pp. 1-430.
Non-Final Office Action in Inter Partes Reexamination Control No. 95/000,659 issued Aug. 16, 2013, 107 pages.
Decision on Petition in Reexamination Control No. 95/000,659 issued Aug. 19, 2013, 3 pages.
Response to Non-Final Office Action in Reexamination Control No. 95/000,659 mailed Oct. 2, 2013 including Exhibits A-C, 37 pages.
Decision on Petition in Reexamination Control No. 95/000,660 issued Jul. 30, 2013, 12 pages.
Non-Final Office Action in Inter Partes Reexamination Control No. 95/000,660 issued Aug. 30, 2013, 23 pages.
RFC: 791. Internet Protocol: DARPA Internet Program Protocol Specification, Sep. 1981, prepared for Defense Advanced Research Projects Agency Information Processing Techniques Office by Information Sciences Institute University of Southern California, 52 pages.

* cited by examiner

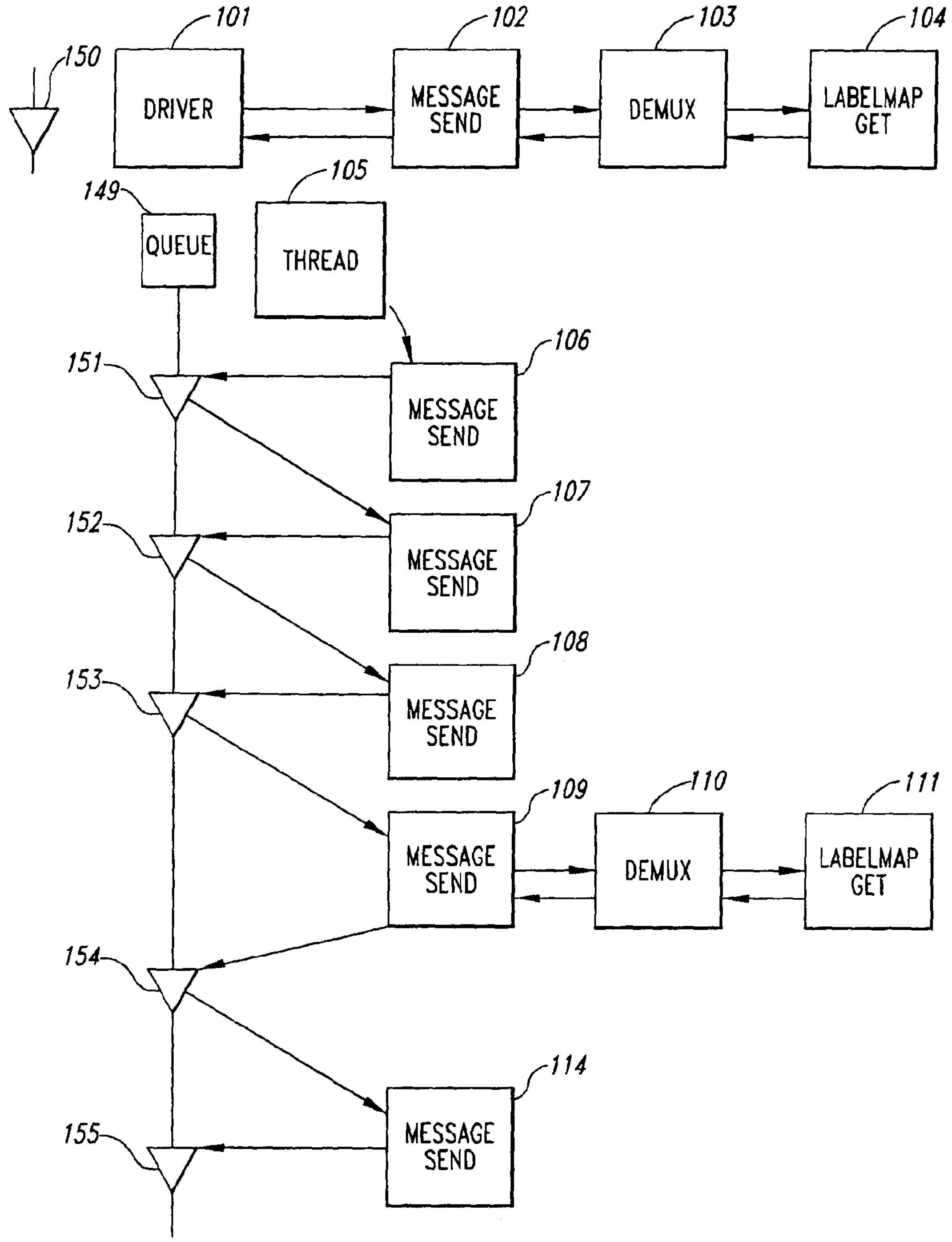

101
DRIVER
102
MESSAGE SEND
103
DEMUX
104
LABELMAP GET
150
149
QUEUE
105
THREAD
106
MESSAGE SEND
151
107
MESSAGE SEND
152
108
MESSAGE SEND
153
109
MESSAGE SEND
110
DEMUX
111
LABELMAP GET
154
114
MESSAGE SEND
155

Fig. 1

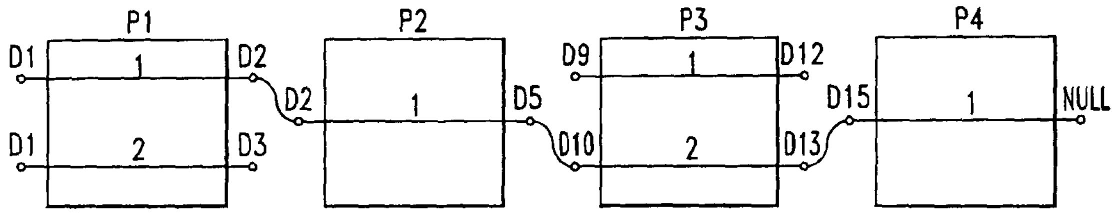

P1
P2
P3
P4
D1
1
D2
D1
2
D3
D2
1
D5
D9
1
D12
D10
2
D13
D15
1
NULL

Fig. 2

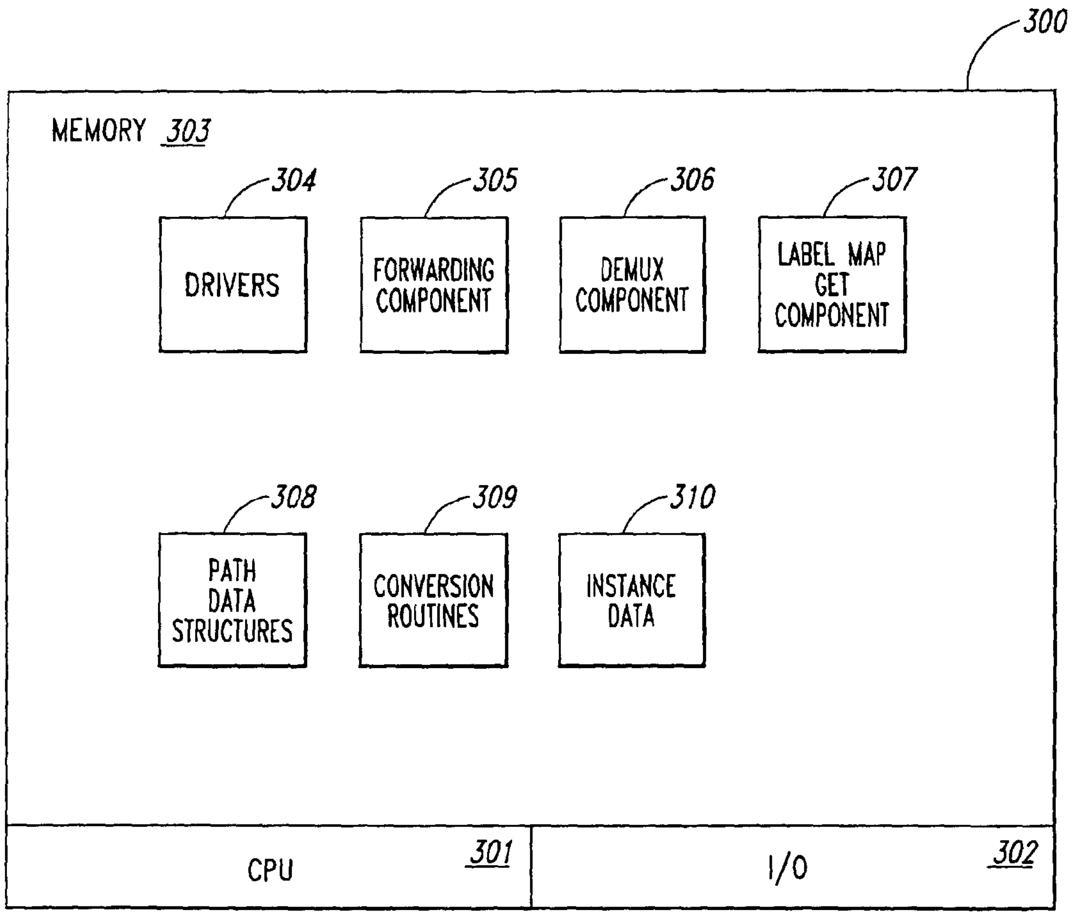

300
MEMORY 303
304
DRIVERS
305
FORWARDING COMPONENT
306
DEMUX COMPONENT
307
LABEL MAP GET COMPONENT
308
PATH DATA STRUCTURES
309
CONVERSION ROUTINES
310
INSTANCE DATA
CPU 301
I/O 302

Fig. 3

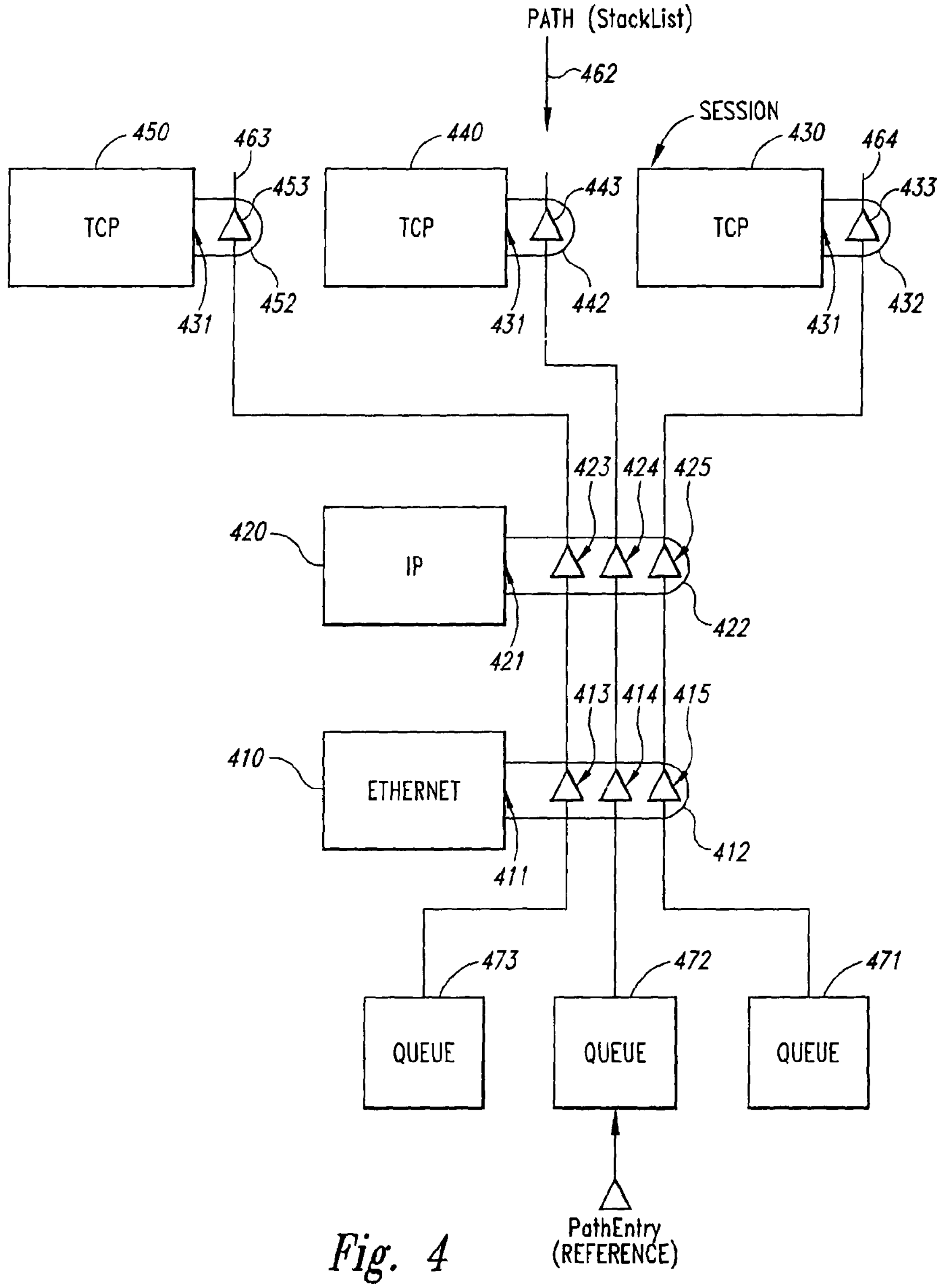
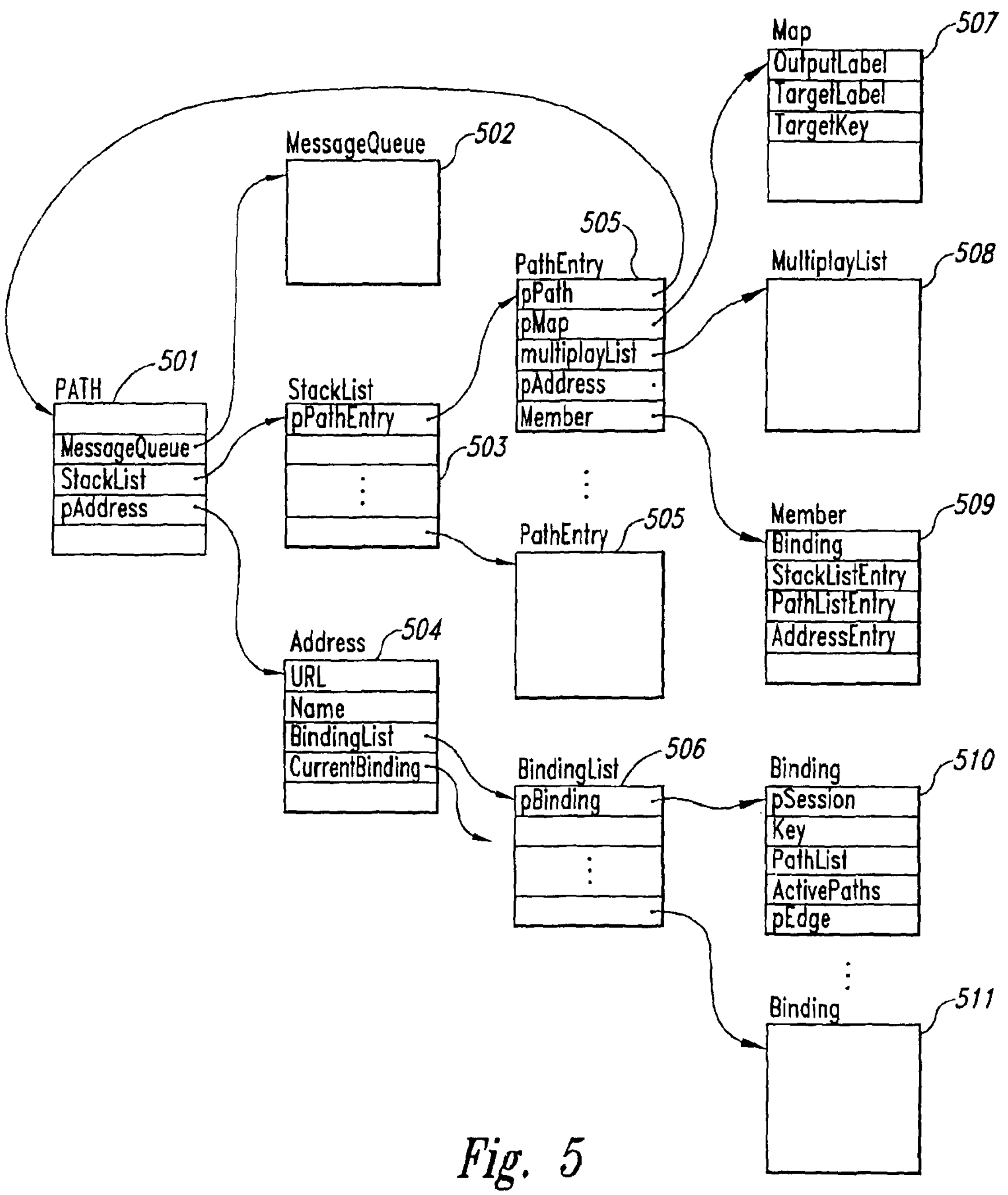

PATH (StackList)
462
SESSION
450
463
440
430
464
453
443
433
TCP
TCP
TCP
431
452
431
442
431
432
423
424
425
420
IP
422
421
413
414
415
410
ETHERNET
412
411
473
472
471
QUEUE
QUEUE
QUEUE
PathEntry
(REFERENCE)

Fig. 4



Map
507
OutputLabel
TargetLabel
TargetKey
MessageQueue
502
505
PathEntry
pPath
pMap
multiplayList
pAddress
Member
MultiplayList
508
PATH
501
MessageQueue
StackList
pAddress
StackList
pPathEntry
503
PathEntry
505
Member
509
Binding
StackListEntry
PathListEntry
AddressEntry
Address
504
URL
Name
BindingList
CurrentBinding
BindingList
506
pBinding
Binding
510
pSession
Key
PathList
ActivePaths
pEdge
Binding
511

Fig. 5

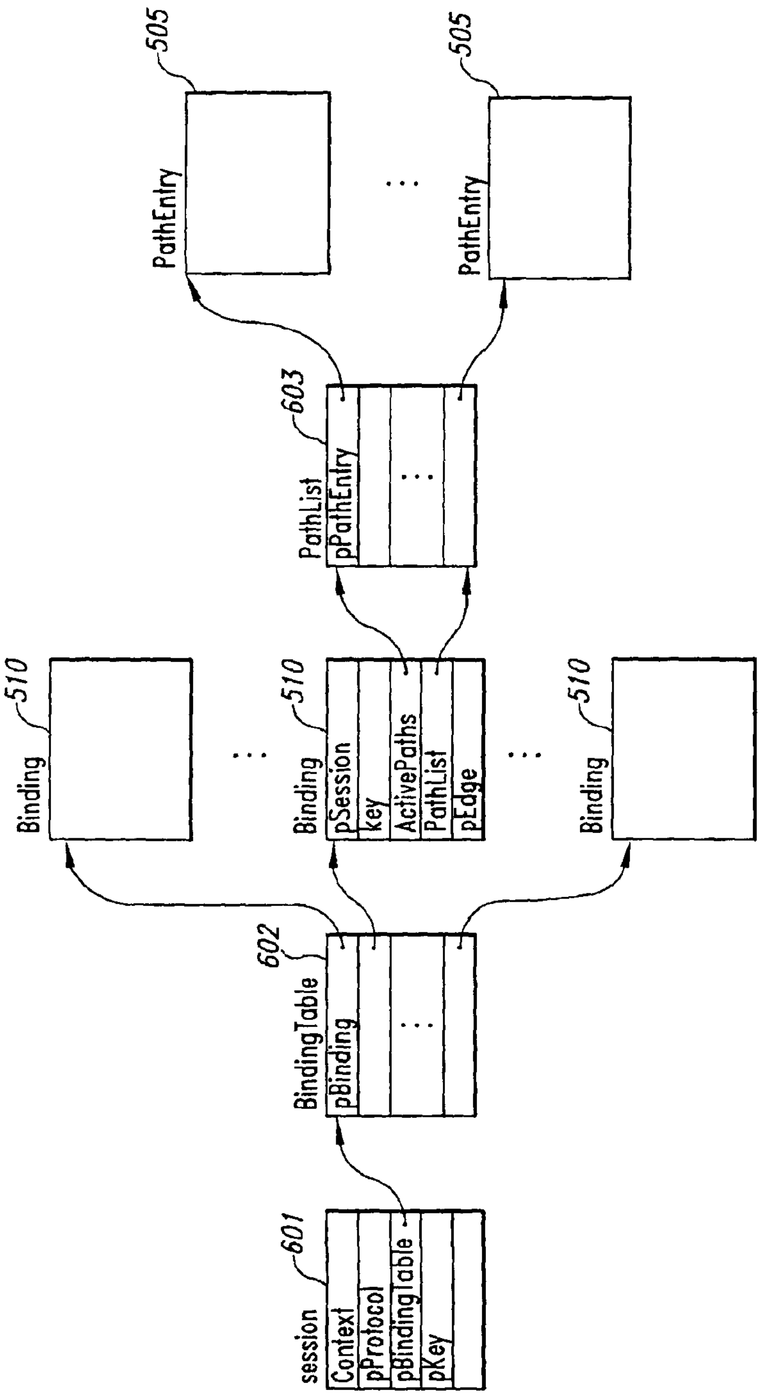

505
PathEntry
505
PathEntry
603
PathList
pPathEntry
510
Binding
510
Binding
pSession
key
ActivePaths
PathList
pEdge
510
Binding
602
BindingTable
pBinding
601
session
Context
pProtocol
pBindingTable
pKey

Fig. 6

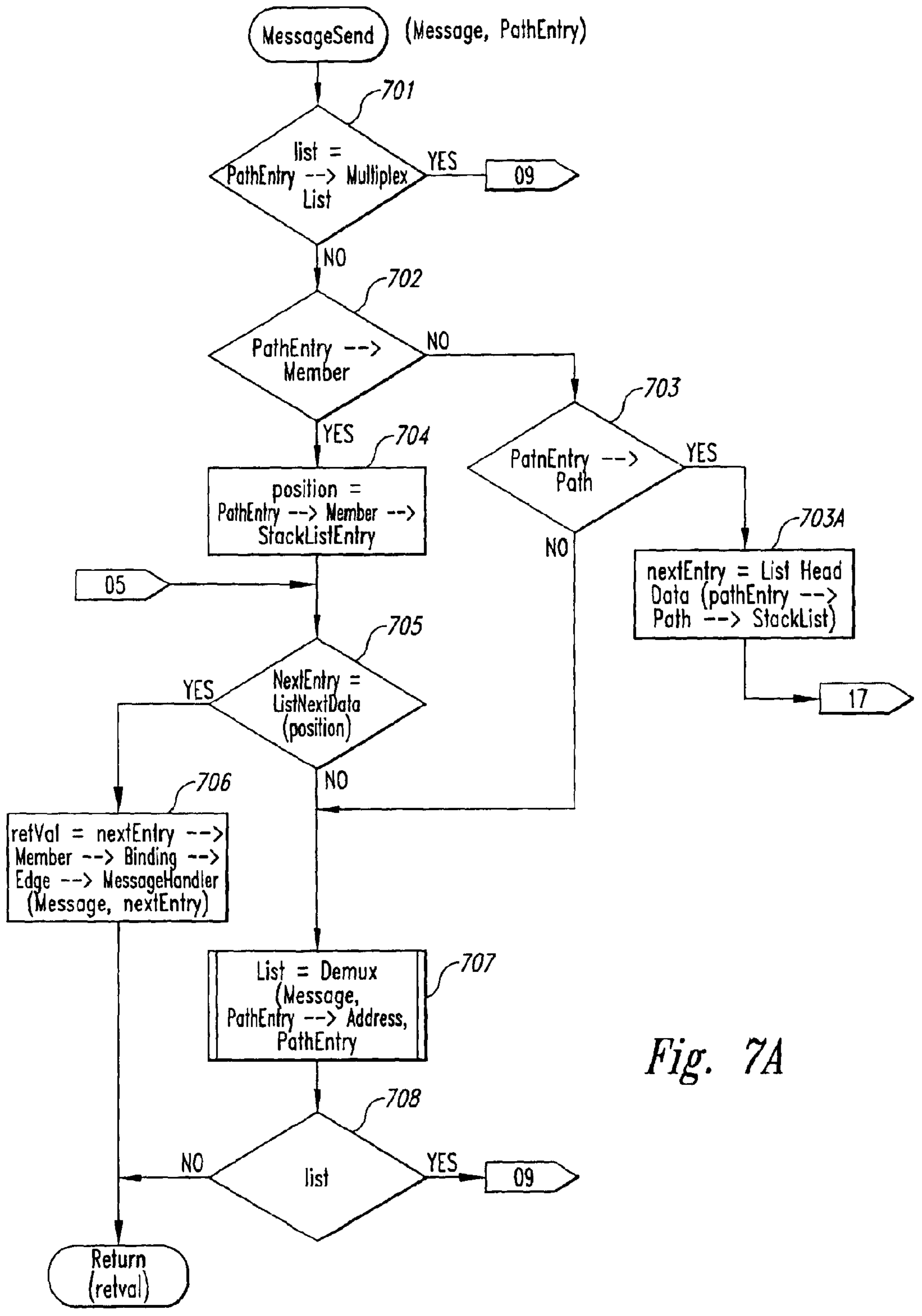

MessageSend
(Message, PathEntry)
701
list = PathEntry --> Multiplex List
YES
09
NO
702
PathEntry --> Member
NO
703
PatnEntry --> Path
YES
703A
nextEntry = List Head Data (pathEntry --> Path --> StackList)
17
NO
YES
704
position = PathEntry --> Member --> StackListEntry
05
705
NextEntry = ListNextData (position)
YES
NO
706
retVal = nextEntry --> Member --> Binding --> Edge --> MessageHandler (Message, nextEntry)
707
List = Demux (Message, PathEntry --> Address, PathEntry
708
list
NO
YES
09
Return (retval)

Fig. 7A

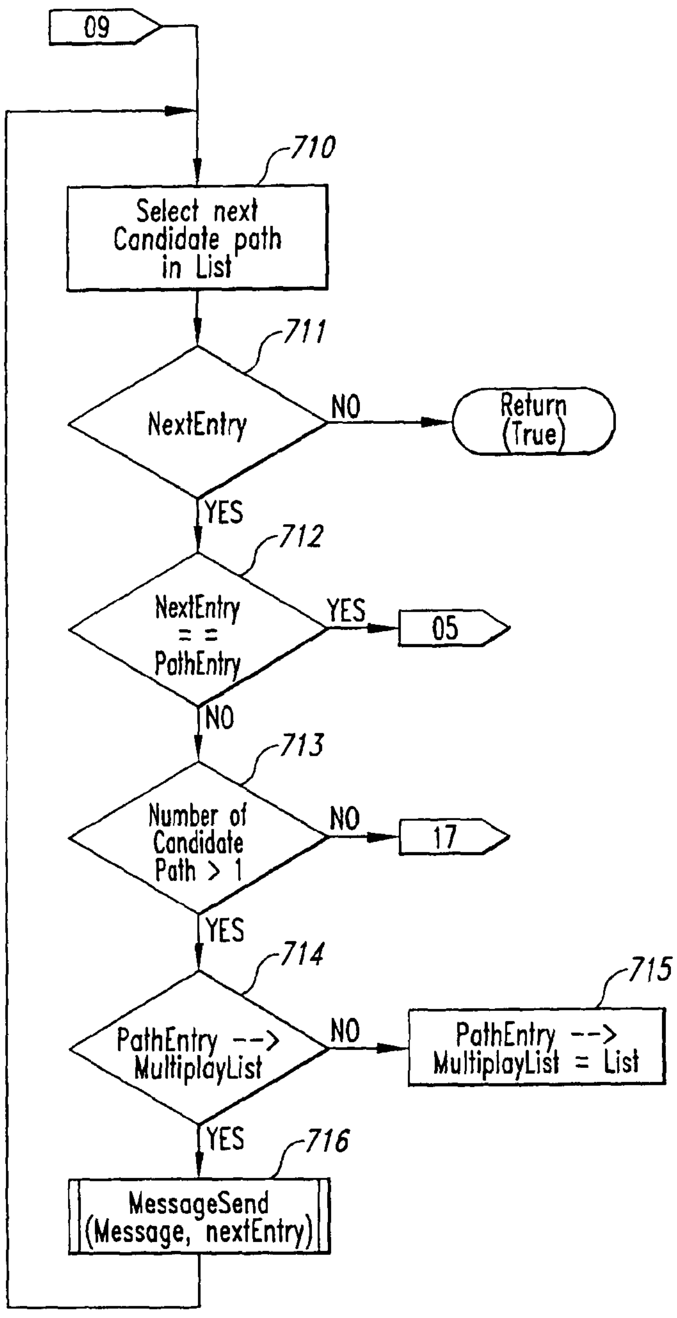

09
710
Select next Candidate path in List
711
NextEntry
NO
Return (True)
YES
712
NextEntry == PathEntry
YES
05
NO
713
Number of Candidate Path > 1
NO
17
YES
714
PathEntry --> MultiplayList
NO
715
PathEntry --> MultiplayList = List
YES
716
MessageSend (Message, nextEntry)

Fig. 7B

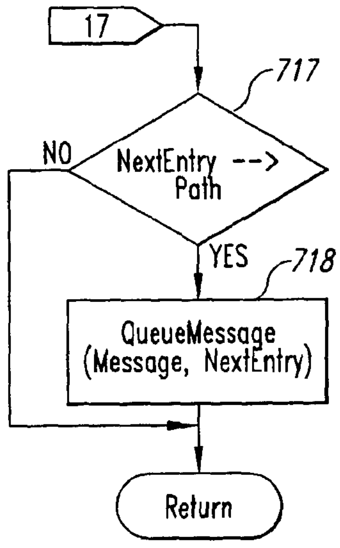

17
717
NextEntry --> Path
NO
YES
718
QueueMessage (Message, NextEntry)
Return

Fig. 7C

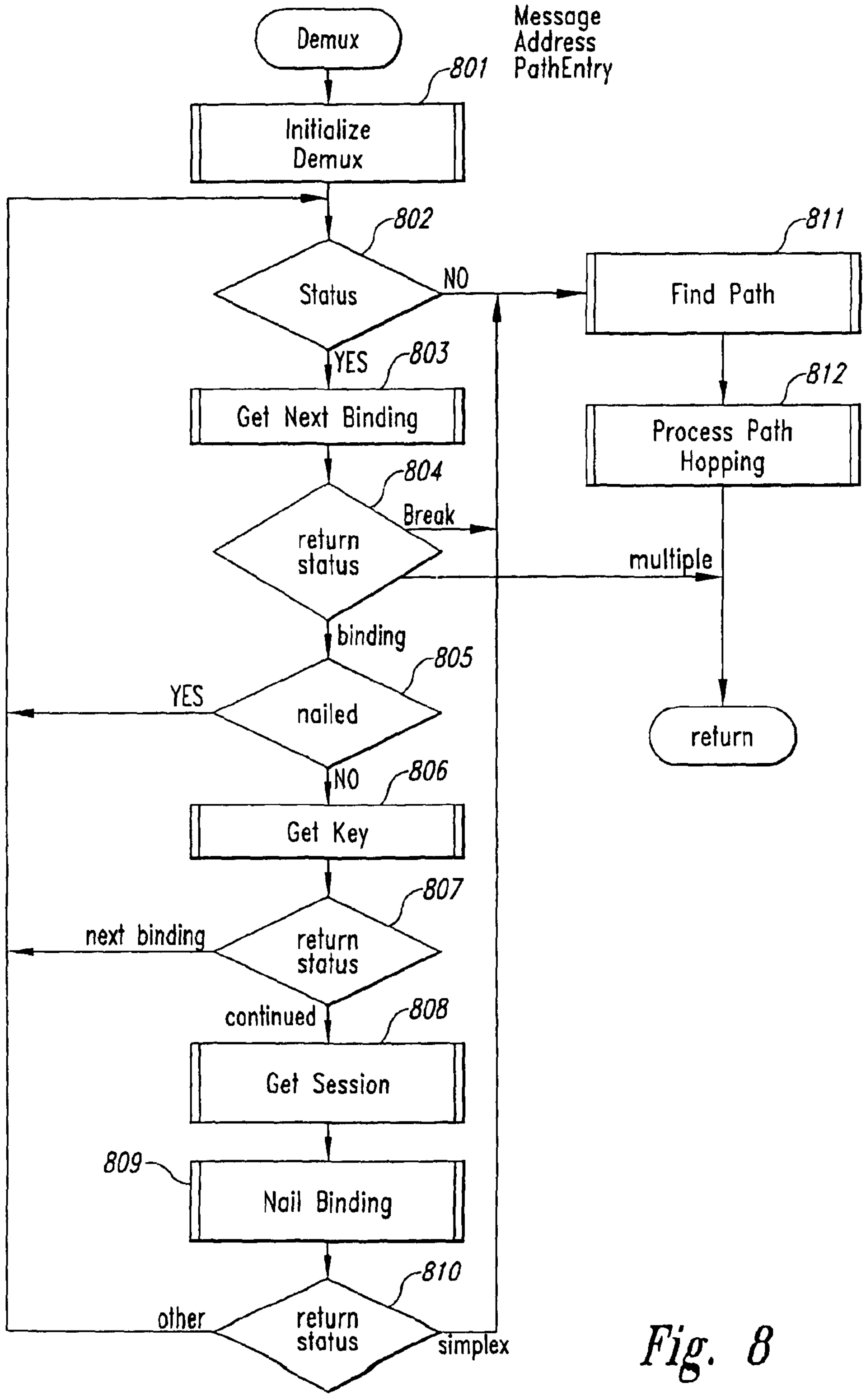
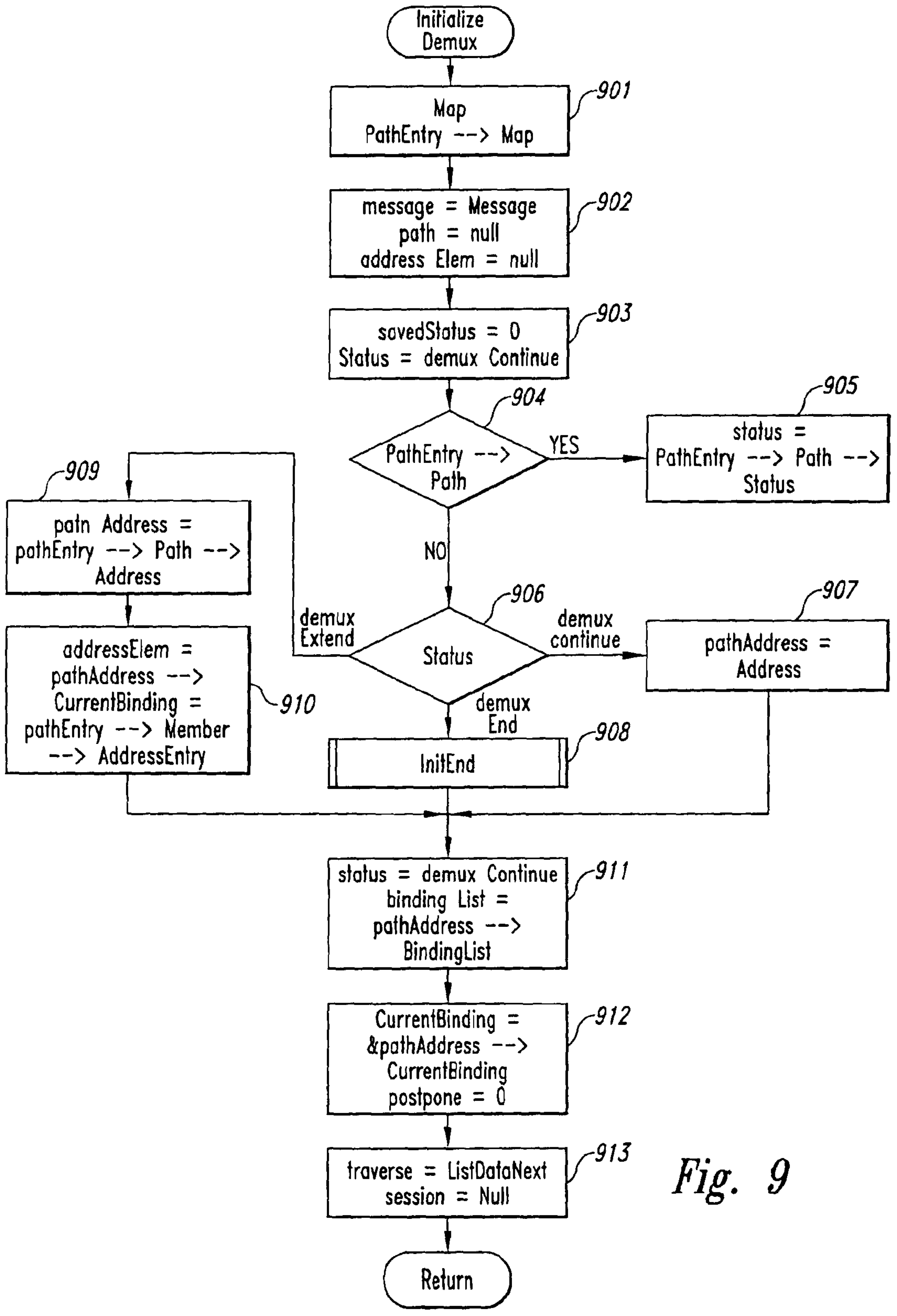

Demux
Message
Address
PathEntry
801
Initialize Demux
802
Status
NO
YES
811
Find Path
803
Get Next Binding
812
Process Path Hopping
804
Break
return status
multiple
binding
805
YES
nailed
return
NO
806
Get Key
807
next binding
return status
continued
808
Get Session
809
Nail Binding
810
other
return status
simplex

Fig. 8



Initialize Demux
Map PathEntry --> Map
901
message = Message path = null address Elem = null
902
savedStatus = 0 Status = demux Continue
903
904
PathEntry --> Path
YES
905
status = PathEntry --> Path --> Status
NO
909
patn Address = pathEntry --> Path --> Address
addressElem = pathAddress --> CurrentBinding = pathEntry --> Member --> AddressEntry
910
demux Extend
906
Status
demux continue
907
pathAddress = Address
demux End
908
InitEnd
status = demux Continue binding List = pathAddress --> BindingList
911
CurrentBinding = &pathAddress --> CurrentBinding postpone = 0
912
traverse = ListDataNext session = Null
913
Return

Fig. 9

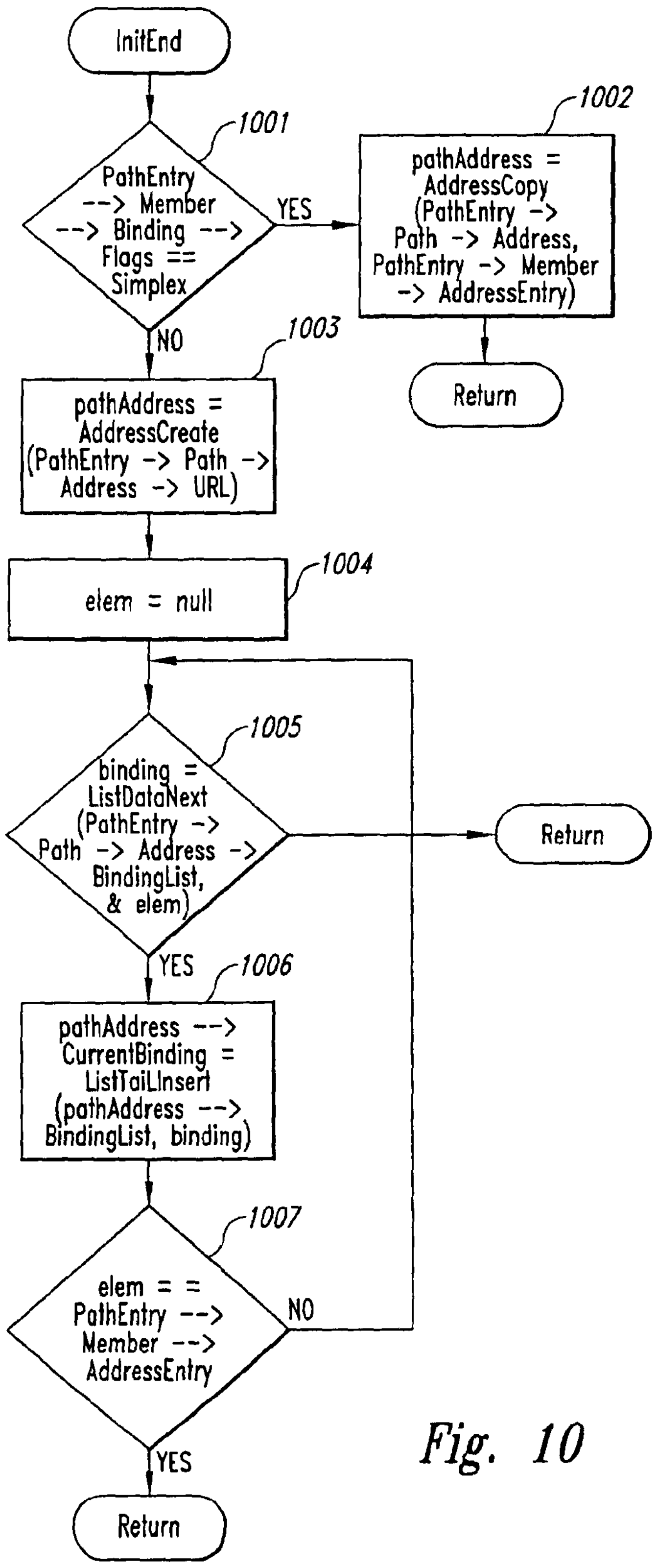

InitEnd
1001
PathEntry --> Member --> Binding --> Flags == Simplex
YES
1002
pathAddress = AddressCopy (PathEntry -> Path -> Address, PathEntry -> Member -> AddressEntry)
Return
NO
1003
pathAddress = AddressCreate (PathEntry -> Path -> Address -> URL)
1004
elem = null
1005
binding = ListDataNext (PathEntry -> Path -> Address -> BindingList, & elem)
Return
YES
1006
pathAddress --> CurrentBinding = ListTailInsert (pathAddress --> BindingList, binding)
1007
elem = = PathEntry --> Member --> AddressEntry
NO
YES
Return

Fig. 10

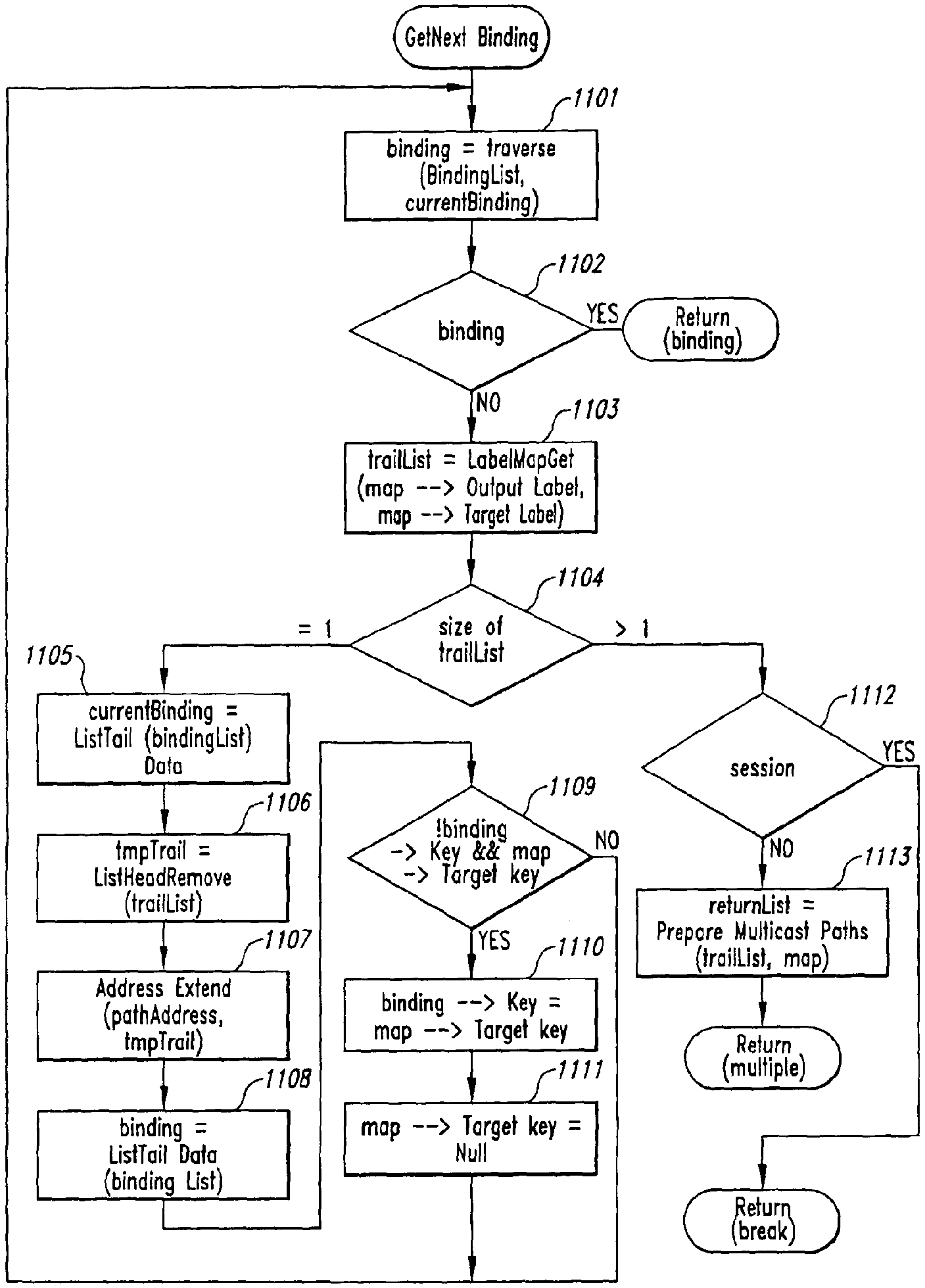
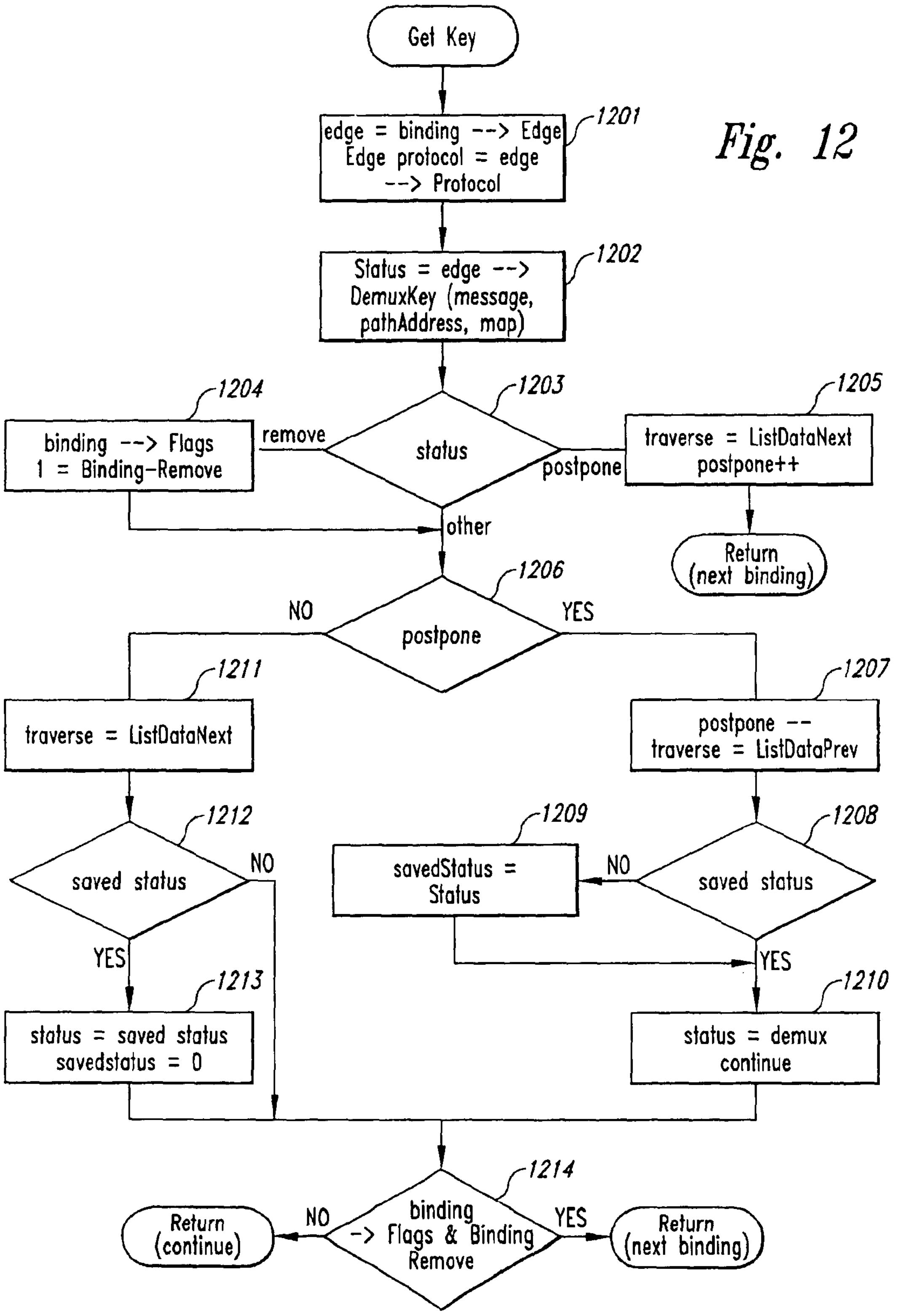

GetNext Binding
1101
binding = traverse (BindingList, currentBinding)
1102
binding
YES
Return (binding)
NO
1103
trailList = LabelMapGet (map --> Output Label, map --> Target Label)
1104
size of trailList
= 1
> 1
1105
currentBinding = ListTail (bindingList) Data
1106
tmpTrail = ListHeadRemove (trailList)
1107
Address Extend (pathAddress, tmpTrail)
1108
binding = ListTail Data (binding List)
1109
!binding -> Key && map -> Target key
NO
YES
1110
binding --> Key = map --> Target key
1111
map --> Target key = Null
1112
session
YES
NO
1113
returnList = Prepare Multicast Paths (trailList, map)
Return (multiple)
Return (break)

Fig. 11



Get Key
edge = binding --> Edge Edge protocol = edge --> Protocol
1201
Status = edge --> DemuxKey (message, pathAddress, map)
1202
status
1203
remove
postpone
other
binding --> Flags 1 = Binding-Remove
1204
traverse = ListDataNext postpone++
1205
Return (next binding)
postpone
1206
NO
YES
traverse = ListDataNext
1211
postpone -- traverse = ListDataPrev
1207
saved status
1212
NO
YES
status = saved status savedstatus = 0
1213
saved status
1208
NO
YES
savedStatus = Status
1209
status = demux continue
1210
binding -> Flags & Binding Remove
1214
NO
YES
Return (continue)
Return (next binding)

Fig. 12

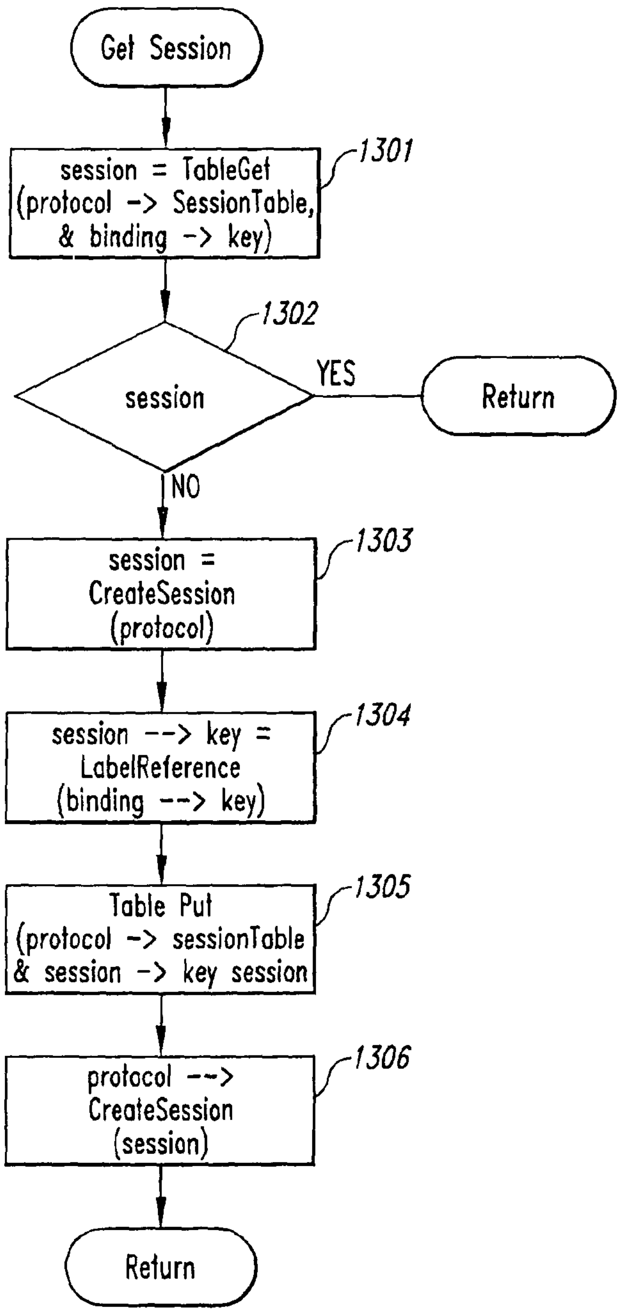

Get Session
session = TableGet (protocol -> SessionTable, & binding -> key)
1301
session
1302
YES
Return
NO
session = CreateSession (protocol)
1303
session --> key = LabelReference (binding --> key)
1304
Table Put (protocol -> sessionTable & session -> key session
1305
protocol --> CreateSession (session)
1306
Return

Fig. 13

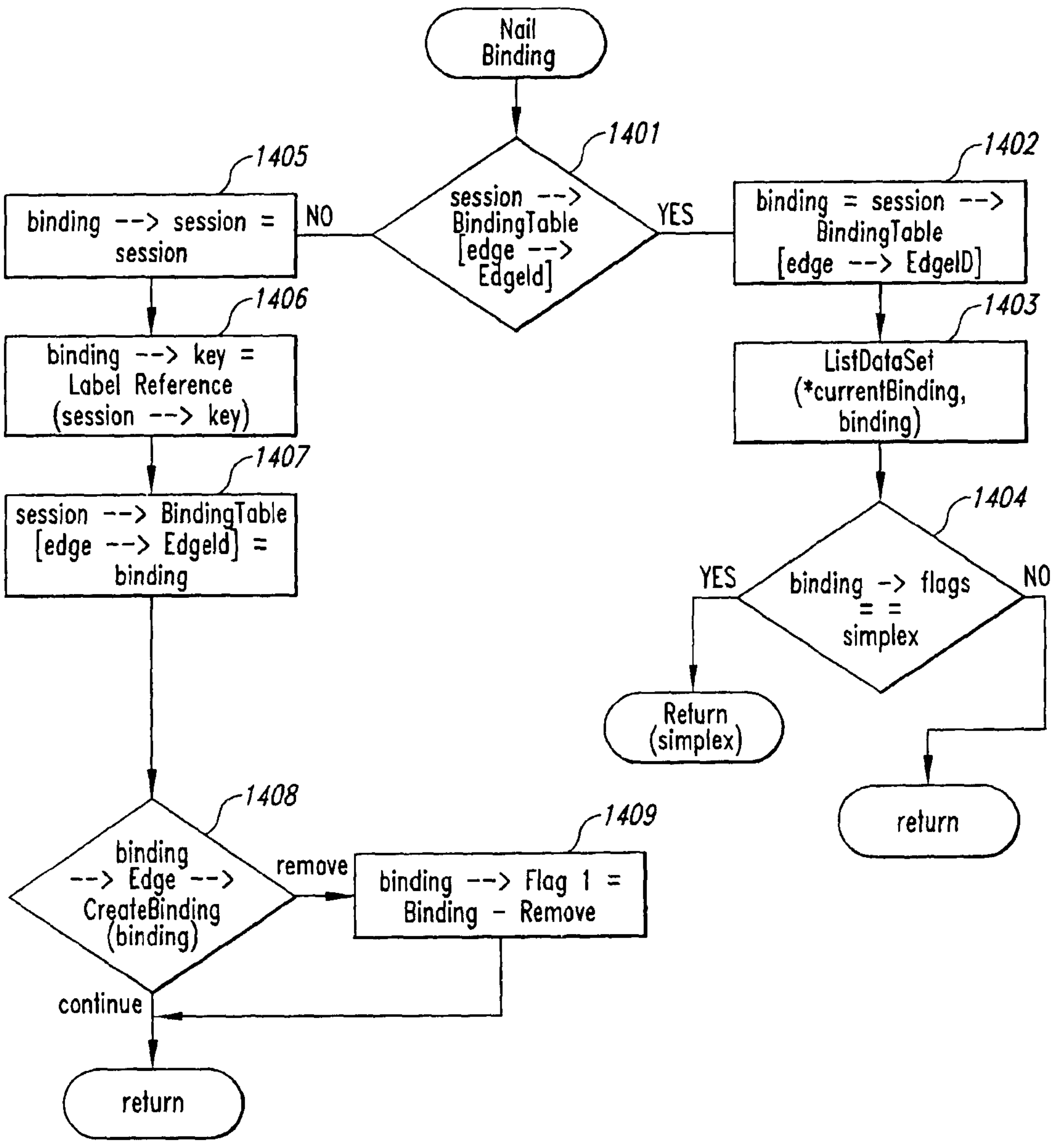

Nail Binding
1401
session --> BindingTable [edge --> EdgeId]
YES
NO
1402
binding = session --> BindingTable [edge --> EdgeID]
1403
ListDataSet (*currentBinding, binding)
1404
binding -> flags = = simplex
YES
NO
Return (simplex)
return
1405
binding --> session = session
1406
binding --> key = Label Reference (session --> key)
1407
session --> BindingTable [edge --> EdgeId] = binding
1408
binding --> Edge --> CreateBinding (binding)
remove
1409
binding --> Flag 1 = Binding - Remove
continue
return

Fig. 14

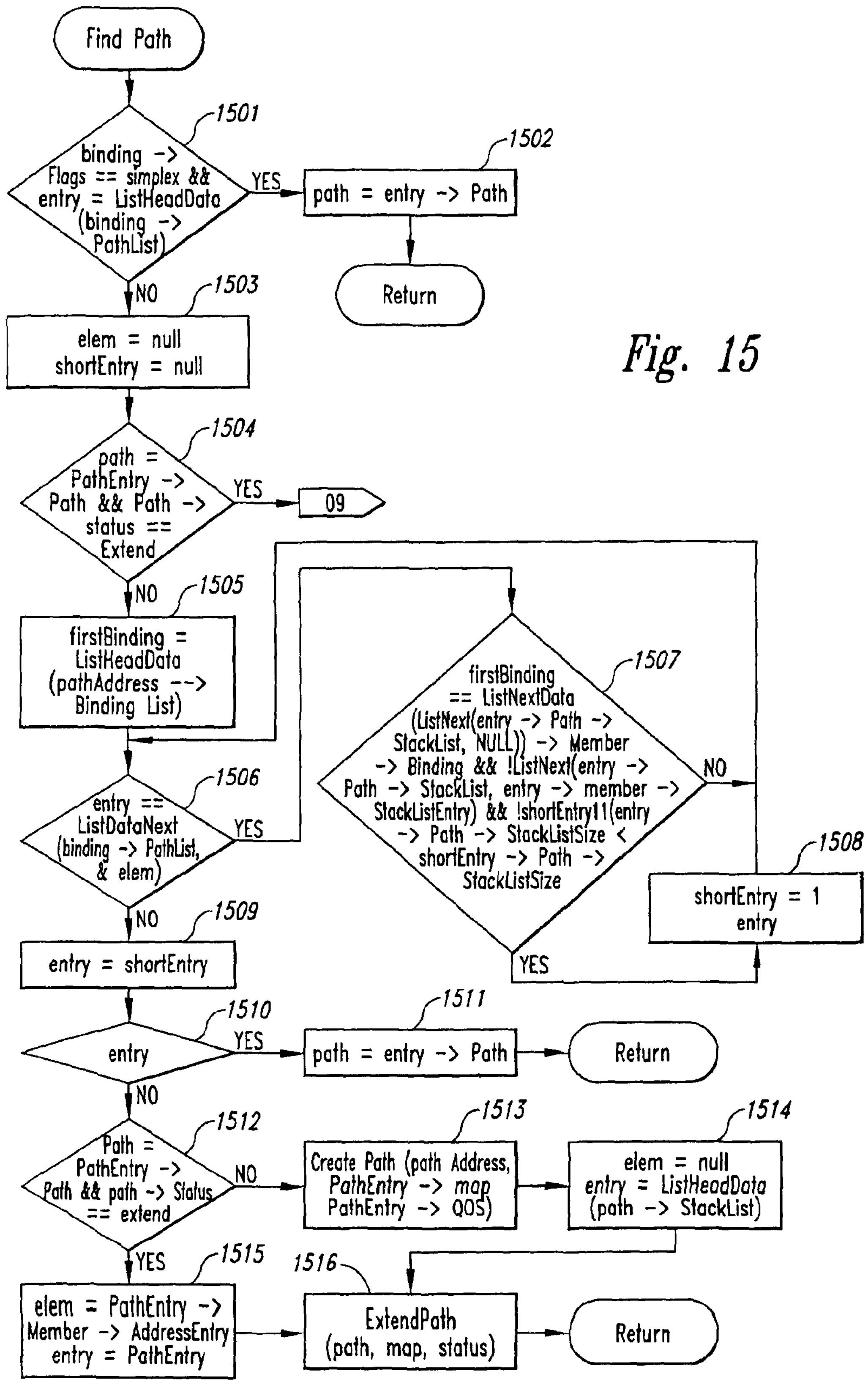

Find Path
1501
binding -> Flags == simplex && entry = ListHeadData (binding -> PathList)
YES
NO
1502
path = entry -> Path
Return
1503
elem = null
shortEntry = null
1504
path = PathEntry -> Path && Path -> status == Extend
09
1505
firstBinding = ListHeadData (pathAddress --> Binding List)
1506
entry == ListDataNext (binding -> PathList, & elem)
1507
firstBinding == ListNextData (ListNext(entry -> Path -> StackList, NULL)) -> Member -> Binding && !ListNext(entry -> Path -> StackList, entry -> member -> StackListEntry) && !shortEntry11(entry -> Path -> StackListSize < shortEntry -> Path -> StackListSize
1508
shortEntry = 1 entry
1509
entry = shortEntry
1510
entry
1511
path = entry -> Path
Return
1512
Path = PathEntry -> Path && path -> Status == extend
1513
Create Path (path Address, PathEntry -> map PathEntry -> QOS)
1514
elem = null
entry = ListHeadData (path -> StackList)
1515
elem = PathEntry -> Member -> AddressEntry
entry = PathEntry
1516
ExtendPath (path, map, status)
Return

Fig. 15

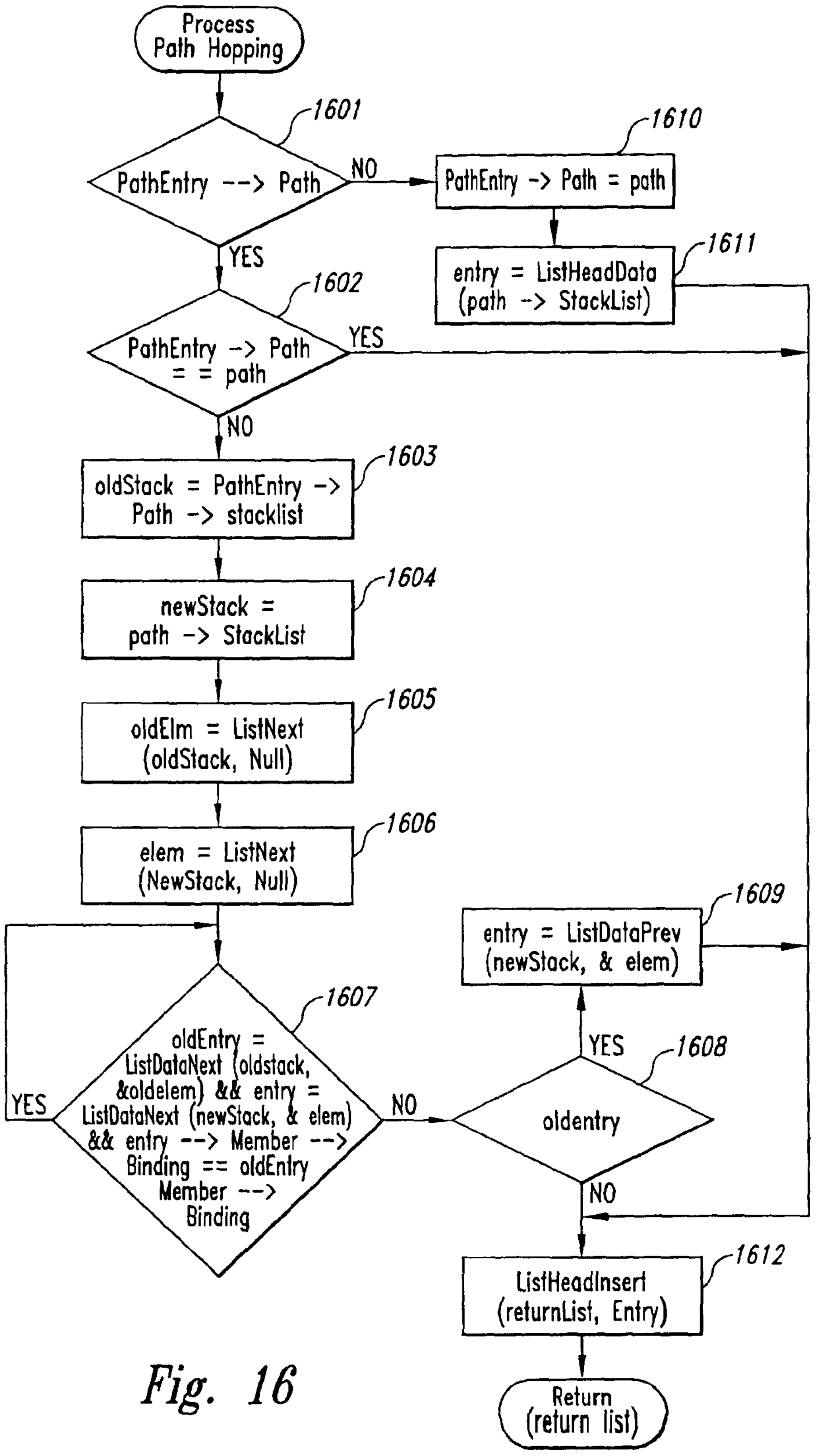

Process Path Hopping
1601
PathEntry --> Path
NO
YES
1610
PathEntry -> Path = path
1611
entry = ListHeadData (path -> StackList)
1602
PathEntry -> Path = = path
YES
NO
1603
oldStack = PathEntry -> Path -> stacklist
1604
newStack = path -> StackList
1605
oldElm = ListNext (oldStack, Null)
1606
elem = ListNext (NewStack, Null)
1607
oldEntry = ListDataNext (oldstack, &oldelem) && entry = ListDataNext (newStack, & elem) && entry --> Member --> Binding == oldEntry Member --> Binding
YES
NO
1608
oldentry
YES
NO
1609
entry = ListDataPrev (newStack, & elem)
1612
ListHeadInsert (returnList, Entry)
Return (return list)

Fig. 16

METHOD AND SYSTEM FOR DATA DEMULTIPLEXING

CROSS REFERENCES TO RELATED APPLICATIONS

The present application is a continuation of U.S. application Ser. No. 13/911,324, filed Jun. 6, 2013 (now U.S. Pat. No. 8,694,683), which is a continuation of U.S. application Ser. No. 13/236,090, filed Sep. 19, 2011 (now abandoned), which is a continuation of U.S. application Ser. No. 10/636,314, filed Aug. 6, 2003 know U.S. Pat. No. 8,055,786), which is a continuation of U.S. application Ser. No. 09/474,664, filed Dec. 29, 1999 (now U.S. Pat. No. 6,629,163); the disclosures of each of the above-referenced applications are incorporated by reference herein in their entireties.

TECHNICAL FIELD

The present invention relates generally to a computer system for data demultiplexing.

BACKGROUND

Computer systems, which are becoming increasingly pervasive, generate data in a wide variety of formats. The Internet is an example of interconnected computer systems that generate data in many different formats. Indeed, when data is generated on one computer system and is transmitted to another computer system to be displayed, the data may be converted in many different intermediate formats before it is eventually displayed. For example, the generating computer system may initially store the data in a bitmap format. To send the data to another computer system, the computer system may first compress the bitmap data and then encrypt the compressed data. The computer system may then convert that compressed data into a TCP format and then into an IP format. The IP formatted data may be converted into a transmission format, such as an ethernet format. The data in the transmission format is then sent to a receiving computer system. The receiving computer system would need to perform each of these conversions in reverse order to convert the data in the bitmap format. In addition, the receiving computer system may need to convert the bitmap data into a format that is appropriate for rendering on output device.

In order to process data in such a wide variety of formats, both sending and receiving computer systems need to have many conversion routines available to support the various formats. These computer systems typically use predefined configuration information to load the correct combination of conversion routines for processing data. These computer systems also use a process-oriented approach when processing data with these conversion routines. When using a process-oriented approach, a computer system may create a separate process for each conversion that needs to take place. A computer system in certain situations, however, can be expected to receive data and to provide data in many different formats that may not be known until the data is received. The overhead of statically providing each possible series of conversion routines is very high. For example, a computer system that serves as a central controller for data received within a home would be expected to process data received via telephone lines, cable TV lines, and satellite connections in many different formats. The central controller would be expected to output the data to computer displays, television displays, entertainment centers, speakers, recording devices, and so on in many different formats. Moreover, since the various conversion routines may

be developed by different organizations, it may not be easy to identify that the output format of one conversion routine is compatible with the input format of another conversion routine.

It would be desirable to have a technique for dynamically identifying a series of conversion routines for processing data. In addition, it would be desirable to have a technique in which the output format of one conversion routine can be identified as being compatible with the input format of another conversion routine. It would also be desirable to store the identification of a series of conversion routines so that the series can be quickly identified when data is received.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a block diagram illustrating example processing of a message by the conversion system.

FIG. **2** is a block diagram illustrating a sequence of edges.

FIG. **3** is a block diagram illustrating components of the conversion system in one embodiment.

FIG. **4** is a block diagram illustrating example path data structures in one embodiment.

FIG. **5** is a block diagram that illustrates the interrelationship of the data structures of a path.

FIG. **6** is a block diagram that illustrates the interrelationship of the data structures associated with a session.

FIGS. **7**A, **7**B, and **7**C comprise a flow diagram illustrating the processing of the message send routine.

FIG. **8** is a flow diagram of the demux routine.

FIG. **9** is a flow diagram of the initialize demux routine.

FIG. **10** is a flow diagram of the init end routine.

FIG. **11** is a flow diagram of a routine to get the next binding.

FIG. **12** is a flow diagram of the get key routine.

FIG. **13** is a flow diagram of the get session routine.

FIG. **14** is a flow diagram of the nail binding routine.

FIG. **15** is a flow diagram of the find path routine.

FIG. **16** is a flow diagram of the process of path hopping routine.

DETAILED DESCRIPTION

A method and system for converting a message that may contain multiple packets from an source format into a target format. When a packet of a message is received, the conversion system in one embodiment searches for and identifies a sequence of conversion routines (or more generally message handlers) for processing the packets of the message by comparing the input and output formats of the conversion routines. (A message is a collection of data that is related in some way, such as stream of video or audio data or an email message.) The identified sequence of conversion routines is used to convert the message from the source format to the target format using various intermediate formats. The conversion system then queues the packet for processing by the identified sequence of conversion routines. The conversion system stores the identified sequence so that the sequence can be quickly found (without searching) when the next packet in the message is received. When subsequent packets of the message are received, the conversion system identifies the sequence and queues the packets for pressing by the sequence. Because the conversion system receives multiple messages with different source and target formats and identifies a sequence of conversion routines for each message, the conversion systems effectively "demultiplexes" the messages. That is, the conversion system demultiplexes the messages by receiving the message, identifying the sequence of conversion routines, and controlling the processing of each message by the identified sequence. Moreover, since the conversion routines may need to retain state information between the receipt of one packet of a message and the next packet of that message, the conversion system maintains state information as an instance or session of the conversion routine. The conversion system routes all packets for a message through the same session of each conversion routine so that the same state or instance information can be used by all packets of the message. A sequence of sessions of conversion routines is referred to as a "path." In one embodiment, each path has a path thread associated with it for processing of each packet destined for that path.

In one embodiment, the packets of the messages are initially received by "drivers," such as an Ethernet driver. When a driver receives a packet, it forwards the packet to a forwarding component of the conversion system. The forwarding component is responsible for identifying the session of the conversion routine that should next process the packet and invoking that conversion routine. When invoked by a driver, the forwarding component may use a demultiplexing ("demux") component to identify the session of the first conversion routine of the path that is to process the packet and then queues the packet for processing by the path. A path thread is associated with each path. Each path thread is responsible for retrieving packets from the queue of its path and forwarding the packets to the forwarding component. When the forwarding component is invoked by a path thread, it initially invokes the first conversion routine in the path. That conversion routine processes the packet and forwards the processed packet to the forwarding component, which then invokes the second conversion routine in the path. The process of invoking the conversion routines and forwarding the processed packet to the next conversion routine continues until the last conversion routine in the path is invoked. A conversion routine may defer invocation of the forwarding component until it aggregates multiple packets or may invoke the forwarding component multiple times for a packet once for each sub-packet.

The forwarding component identifies the next conversion routine in the path using the demux component and stores that identification so that the forwarding component can quickly identify the conversion routine when subsequent packets of the same message are received. The demux component, searches for the conversion routine and session that is to next process a packet. The demux component then stores the identification of the session and conversion routine as part of a path data structure so that the conversion system does not need to search for the session and conversion routine when requested to demultiplex subsequent packets of the same message. When searching for the next conversion routine, the demux component invokes a label map get component that identifies the next conversion routine. Once the conversion routine is found, the demux component identifies the session associated with that message by, in one embodiment, invoking code associated with the conversion routine. In general, the code of the conversion routine determines what session should be associated with a message. In certain situations, multiple messages may share the same session. The demux component then extends the path for processing that packet to include that session and conversion routine. The sessions are identified so that each packet is associated with the appropriate state information. The dynamic identification of conversion routines is described in U.S. patent application Ser. No. 11,933,093, filed on Oct. 31, 2007 (now U.S. Pat. No. 7,730, 211), entitled "Method and System for Generating a Mapping Between Types of Data," which is hereby incorporated by reference.

FIG. **1** is a block diagram illustrating example processing of a message by the conversion system. The driver **101** receives the packets of the message from a network. The driver performs any appropriate processing of the packet and invokes a message send routine passing the processed packet along with a reference path entry **150**. The message send routine is an embodiment of the forwarding component. A path is represented by a series of path entries, which are represented by triangles. Each member path entry represents a session and conversion routine of the path, and a reference path entry represents the overall path. The passed reference path entry **150** indicates to the message send routine that it is being invoked by a driver. The message send routine invokes the demux routine **102** to search for and identify the path of sessions that is to process the packet. The demux routine may in turn invoke the label map get routine **104** to identify a sequence of conversion routines for processing the packet. In this example, the label map get routine identifies the first three conversion routines, and the demux routine creates the member path entries **151**, **152**, **153** of the path for these conversion routines. Each path entry identifies a session for a conversion routine, and the sequence of path entries **151-155** identifies a path. The message send routine then queues the packet on the queue **149** for the path that is to process the packets of the message. The path thread **105** for the path retrieves the packet from the queue and invokes the message send routine **106** passing the packet and an indication of the path. The message send routine determines that the next session and conversion routine as indicated by path entry **151** has already been found. The message send routine then invokes the instance of the conversion routine for the session. The conversion routine processes the packet and then invokes the message send routine **107**. This processing continues until the message send routine invokes the demux routine **110** after the packet is processed by the conversion routine represented by path entry **153**. The demux routine examines the path and determines that it has no more path entries. The demux routine then invokes the label map get routine **111** to identify the conversion routines for further processing of the packet. When the conversion routines are identified, the demux routine adds path entries **154**, **155** to the path. The messages send routine invokes the conversion routine associated with path entry **154**. Eventually, the conversion routine associated with path entry **155** performs the final processing for the path.

The label map get routine identifies a sequence of "edges" for converting data in one format into another format. Each edge corresponds to a conversion routine for converting data from one format to another. Each edge is part of a "protocol" (or more generally a component) that may include multiple related edges. For example, a protocol may have edges that each convert data in one format into several different formats. Each edge has an input format and an output format. The label map get routine identifies a sequence of edges such that the output format of each edge is compatible with the input format of another edge in the sequence, except for the input format of the first edge in the sequence and the output format of the last edge in the sequence. FIG. **2** is a block diagram illustrating a sequence of edges. Protocol PI includes an edge for converting format D**1** to format D**2** and an edge for converting format D**1** to format D**3**; protocol P**2** includes an edge for converting format D**2** to format D**5**, and so on. A 30 sequence for converting format D**1** to format D**15** is shown by the curved lines and is defined by the address "P**1**:I, P**2**:**1**, P**3**:**2**, P**4**:**7**." When a packet of data in format DI is processed by this sequence, it is converted to format DIS. During the process, the packet of data is sequentially converted to format D**2**, D**5**, and D**13**. The output format of protocol P**2**, edge **1** (i.e., P**2**:**1**) is format D**5**, but the input format of P**3**:**2** is format D**10**. The label map get routine uses an aliasing mechanism by which two formats, such as D**5** and D**10** are identified as being compatible. The use of aliasing allows different names of the same format or compatible formats to be correlated.

FIG. **3** is a block diagram illustrating components of the conversion system in one embodiment. The conversion system **300** can operate on a computer system with a central processing unit **301**, I/O devices **302**, and memory **303**. The **110** devices may include an Internet connection, a connection to various output devices such as a television, and a connection to various input devices such as a television receiver. The media mapping system may be stored as instructions on a computer-readable medium, such as a disk drive, memory, or data transmission medium. The data structures of the media mapping system may also be stored on a computer-readable medium. The conversion system includes drivers **304**, a forwarding component **305**, a demux component **306**, a label map get component **307**, path data structures **308**, conversion routines **309**, and instance data **310**. Each driver receives data in a source format and forwards the data to the forwarding component. The forwarding component identifies the next conversion routine in the path and invokes that conversion routine to process a packet. The forwarding component may invoke the demux component to search for the next conversion routine and add that conversion routine to the path. The demux component may invoke the label map get component to identify the next conversion routine to process the packet. The demux component stores information defining the paths in the path structures. The conversion routines store their state information in the instance data.

FIG. **4** is a block diagram illustrating example path data structures in one embodiment. The demux component identifies a sequence of "edges" for converting data in one format into another format by invoking the label map get component. Each edge corresponds to a conversion routine for converting data from one format to another. As discussed above, each edge is part of a "protocol" that may include multiple related edges. For example, a protocol may have edges that each convert data in one format into several different formats. Each edge has as an input format ("input label") and an output format ("output label"). Each rectangle represents a session **410**, **420**, **430**, **440**, **450** for a protocol. A session corresponds to an instance of a protocol. That is, the session includes the protocol and state information associated with that instance of the protocol. Session **410** corresponds to a session for an Ethernet protocol; session **420** corresponds to a session for an IP protocol; and sessions **430**, **440**, **450** correspond to sessions for a TCP protocol. FIG. **4** illustrates three paths **461**, **462**, **463**. Each path includes edges **411**, **421**, **431**. The paths share the same Ethernet session **410** and IP session **420**, but each path has a unique TCP session **430**, **440**, **450**. Thus, path **461** includes sessions **410**, **420**, and **430**; path **462** includes sessions **410**, **420**, and **440**; and path **463** includes sessions **410**, **420**, and **450**. The conversion system represents each path by a sequence of path entry structures. Each path entry structure is represented by a triangle. Thus, path **461** is represented by path entries **415**, **425**, and **433**. The conversion system represents the path entries of a path by a stack list. Each path also has a queue **471**, **472**, **473** associated with it. Each queue stores the messages that are to be processed by the conversion routines of the edges of the path. Each session includes a binding **412**, **422**, **432**, **442**, **452** that is represented by an oblong shape adjacent to the corresponding edge. A binding for an edge of a session represents those paths that include the edge. The binding **412** indicates that three paths are bound (or "nailed") to edge **411** of the Ethernet session **410**. The conversion system uses a path list to track the paths that are bound to a binding. The path list of binding **412** identifies path entries **413**, **414**, and **415**.

FIG. **5** is a block diagram that illustrates the interrelationship of the data structures of a path. Each path has a corresponding path structure **501** that contains status information and pointers to a message queue structure **502**, a stack list structure **503**, and a path address structure **504**. The status of a path can be extend, continue, or end. Each message handler returns a status for the path. The status of extend means that additional path entries should be added to the path. The status of end means that this path should end at this point and subsequent processing should continue at a new path. The status of continue means that the protocol does not care how the path is handled. In one embodiment, when a path has a status of continue, the system creates a copy of the path and extends the copy. The message queue structure identifies the messages (or packets of a message) that are queued up for processing by the path and identifies the path entry at where the processing should start. The stack list structure contains a list of pointers to the path entry structures **505** that comprise the path. Each path entry structure contains a pointer to the corresponding path data structure, a pointer to a map structure **507**, a pointer to a multiplex list **508**, a pointer to the corresponding path address structure, and a pointer to a member structure **509**. A map structure identifies the output label of the edge of the path entry and optionally a target label and a target key. A target key identifies the session associated with the protocol that converts the packet to the target label. (The terms "media," "label," and "format" are used interchangeably to refer to the output of a protocol.) The multiplex list is used during the demux process to track possible next edges when a path is being identified as having more than one next edge. The member structure indicates that the path entry represents an edge of a path and contains a pointer to a binding structure to which the path entry is associated (or "nailed"), a stack list entry is the position of the path entry within the associated stack list, a path list entry is the position of the path entry within the associated path list of a binding and an address entry is the position of the binding within the associated path address. A path address of a path identifies the bindings to which the path entries are bound. The path address structure contains a URL for the path, the name of the path identified by the address, a pointer to a binding list structure **506**, and the identification of the current binding within the binding list. The URL (e.g., "protocol://tcp(0)/ip(0)/eth(0)") identifies conversion routines (e.g., protocols and edges) of a path in a human-readable format. The URL (universal resource locator) includes a type field (e.g., "protocol") followed by a sequence of items (e.g., "tcp(0)"). The type field specifies the format of the following information in the URL, that specifies that the type field is followed by a sequence of items. Each item identifies a protocol and an edge (e.g., the protocol is "tcp" and the edge is "0"). In one embodiment, the items of a URL may also contain an identifier of state information that is to be used when processing a message. These URLs can be used to illustrate to a user various paths that are available for processing a message. The current binding is the last binding in the path as the path is being built. The binding list structure contains a list of pointers to the binding structures associated with the path. Each binding structure **510** contains a pointer to a session structure, a pointer to an edge structure, a key, a path list structure, and a list of active paths through the binding. The key identifies the state information for a session of a protocol. A path list structure contains pointers to the path entry structures associated with the binding.

FIG. **6** is a block diagram that illustrates the interrelationship of the data structures associated with a session. A session structure **601** contains the context for the session, a pointer to a protocol structure for the session, a pointer to a binding table structure **602** for the bindings associated with the session, and the key. The binding table structure contains a list of pointers to the binding structures **510** for the session. The binding structure is described above with reference to FIG. **5**. The path list structure **603** of the binding structure contains a list of pointers to path entry structures **505**. The path entry structures are described with reference to FIG. **5**.

FIGS. **7**A, **7**B, and **7**C comprise a flow diagram illustrating the processing of the message send routine. The message send routine is passed a message along with the path entry associated with the session that last processed the message. The message send routine invokes the message handler of the next edge in the path or queues the message for processing by a path. The message handler invokes the demux routine to identify the next path entry of the path. When a driver receives a message, it invokes the message send routine passing a reference path entry. The message send routine examines the passed path entry to determine (1) whether multiple paths branch from the path of the passed path entry, (2) whether the passed path entry is a reference with an associated path, or (3) whether the passed path entry is a member with a next path entry. If multiple paths branch from the path of the passed path entry, then the routine recursively invokes the message send routine for each path. If the path entry is a reference with an associated path, then the driver previously invoked the message send routine, which associated a path with the reference path entry, and the routine places the message on the queue for the path. If the passed path entry is a member with a next path entry, then the routine invokes the message handler (i.e., conversion routine of the edge) associated with the next path entry. If the passed path entry is a reference without an associated path or is a member without a next path entry, then the routine invokes the demux routine to identify the next path entry. The routine then recursively invokes the messages send routine passing that next path entry. In decision block **701**, if the passed path entry has a multiplex list, then the path branches off into multiple paths and the routine continues at block **709**, else the routine continues at block **702**. A packet may be processed by several different paths. For example, if a certain message is directed to two different output devices, then the message is processed by two different paths. Also, a message may need to be processed by multiple partial paths when searching for a complete path. In decision block **702**, if the passed path entry is a member, then either the next path entry indicates a nailed binding or the path needs to be extended and the routine continues at block **704**, else the routine continues at block **703**. A nailed binding is a binding (e.g., edge and protocol) is associated with a session. In decision block **703**, the passed path entry is a reference and if the passed path entry has an associated path, then the routine can queue the message for the associated path and the routine continues at block **703**A, else the routine needs to identify a path and the routine continues at block **707**. In block **703**A, the routine sets the entry to the first path entry in the path and continues at block **717**. In block **704**, the routine sets the variable position to the stack list entry of the passed path entry. In decision block **705**, the routine sets the variable next entry to the next path entry in the path. If there is a next entry in the path, then the next session and edge of the protocol have been identified and the routine continues at block **706**, else the routine continues at block **707**. In block **706**, the routine passes the message to the message handler of the edge associated with the next entry and then returns. In block **706**, the routine invokes the demux routine passing the passed message, the address of the passed path entry, and the passed path entry. The demux routine returns a list of candidate paths for processing of the message. In decision block **708**, if at least one candidate path is returned, then the routine continues at block **709**, else the routine returns.

Blocks **709-716** illustrate the processing of a list of candidate paths that extend from the passed path entry. In blocks **710-716**, the routine loops selecting each candidate path and sending the message to be process by each candidate path. In block **710**, the routine sets the next entry to the first path entry of the next candidate path. In decision block **711**, if all the candidate paths have not yet been processed, then the routine continues at block **712**, else the routine returns. In decision block **712**, if the next entry is equal to the passed path entry, then the path is to be extended and the routine continues at block **705**, else the routine continues at block **713**. The candidate paths include a first path entry that is a reference path entry for new paths or that is the last path entry of a path being extended. In decision block **713**, if the number of candidate paths is greater than one, then the routine continues at block **714**, else the routine continues at block **718**. In decision block **714**, if the passed path entry has a multiplex list associated with it, then the routine continues at block **716**, else the routine continues at block **715**. In block **715**, **11** the routine associates the list of candidate path with the multiplex list of the passed path entry and continues at block **716**. In block **716**, the routine sends the message to the next entry by recursively invoking the message send routine. The routine then loops to block **710** to select the next entry associated with the next candidate path.

Blocks **717-718** are performed when the passed path entry is a reference path entry that has a path associated with it. In block **717**, if there is a path associated with the next entry, then the routine continues at block **718**, else the routine returns. In block **718**, the routine queues the message for the path of the next entry and then returns.

FIG. **8** is a flow diagram of the demux routine. This routine is passed the packet (message) that is received, an address structure, and a path entry structure. The demux routine extends a path, creating one if necessary. The routine loops identifying the next binding (edge and protocol) that is to process the message and "nailing" the binding to a session for the message, if not already nailed. After identifying the nailed binding, the routine searches for the shortest path through the nailed binding, creating a path if none exists. In block **801**, the routine invokes the initialize demux routine. In blocks **802-810**, the routine loops identifying a path or portion of a path for processing the passed message. In decision block **802**, if there is a current status, which was returned by the demuxkey routine that was last invoked (e.g., continue, extend, end, or postpone), then the routine continues at block **803**, else the routine continues at block **811**. In block **803**, the routine invokes the get next binding routine. The get next binding routine returns the next binding in the path. The binding is the edge of a protocol. That routine extends the path as appropriate to include the binding. The routine returns a return status of break, binding, or multiple. The return status of binding indicates that the next binding in the path was found by extending the path as appropriate and the routine continues to "nail" the binding to a session as appropriate. The return status of multiple means that multiple trails (e.g., candidate paths) were identified as possible extensions of the path. In a decision block **804**, if the return status is break, then the routine continues at block **811**. If the return status is multiple, then the routine returns. If the return status is binding, then the routine continues at block **805**. In decision block **805**, if the retrieved binding is nailed as indicated by being assigned to a session, then the routine loops to block **802**, else the routine continues at block **806**. In block **806**, the routine invokes the get key routine of the edge associated with the binding. The get key routine creates the key for the session associated with the message. If a key cannot be created until subsequent bindings are processed or because the current binding is to be removed, then the get key routine returns a next binding status, else it returns a continue status. In decision block **807**, if the return status of the get key routine is next binding, then the routine loops to block **802** to get the next binding, else the routine continues at block **808**. In block **808**, the routine invokes the routine get session. The routine get session returns the session associated with the key, creating a new session if necessary. In block **809**, the routine invokes the routine nail binding. The routine nail binding retrieves the binding if one is already nailed to the session. Otherwise, that routine nails the binding to the session. In decision block **810**, if the nail binding routine returns a status of simplex, then the routine continues at block **811** because only one path can use the session, else the routine loops to block **802**. Immediately upon return from the nail binding routine, the routine may invoke a set map routine of the edge passing the session and a map to allow the edge to set its map. In block **811**, the routine invokes the find path routine, which finds the shortest path through the binding list and creates a path if necessary. In block **812**, the routine invokes the process path hopping routine, which determines whether the identified path is part of a different path. Path hopping occurs when, for example, IP fragments are built up along separate paths, but once the fragments are built up they can be processed by the same subsequent path.

FIG. **9** is a flow diagram of the initialize demux routine. This routine is invoked to initialize the local data structures that are used in the demux process and to identify the initial binding. The demux routine finds the shortest path from the initial binding to the final binding. If the current status is demux extend, then the routine is to extend the path of the passed path entry by adding additional path entries. If the current status is demux end, then the demux routine is ending the current path. If the current status is demux continue, then the demux routine is in the process of continuing to extend or in the process of starting a path identified by the passed address. In block **901**, the routine sets the local map structure to the map structure in the passed path entry structure. The map structure identifies the output label, the target label, and the target key. In the block **902**, the routine initializes the local message structure to the passed message structure and initializes the pointers path and address element to null. In block **903**, the routine sets of the variable saved status to 0 and the variable status to demux continue. The variable saved status is used to track the status of the demux process when backtracking to nail a binding whose nail was postponed. In decision block **904**, if the passed path entry is associated with a path, then the routine continues at block **905**, else the routine continues at block **906**. In block **905**, the routine sets the variable status to the status of that path. In block **906**, if the variable status is demux continue, then the routine continues at block **907**. If the variable status is demux end, then the routine continues at block **908**. If the variable status is demux extend, then the routine continues at block **909**. In block **907**, the status is demux continue, and the routine sets the local pointer path address to the passed address and continues at block **911**. In block **908**, the status is demux end, and the routine invokes the init end routine and continues at block **911**. In block **909**, the status is demux extend, and the routine sets the local path address to the address of the path that contains the passed path entry. In block **910**, the routine sets the address element and the current binding of the path address pointed to by the local pointer path address to the address entry of the member structure of the passed path entry. In the block **911**, the routine sets the local variable status to demux continue and sets the local binding list structure to the binding list structure from the local path address structure. In block **912**, the routine sets the local pointer current binding to the address of the current binding pointed to by local pointer path address and sets the local variable postpone to 0. In block **913**, the routine sets the function traverse to the function that retrieves the next data in a list and sets the local pointer session to null. The routine then returns.

FIG. **10** is a flow diagram of the init end routine. If the path is simplex, then the routine creates a new path from where the other one ended, else the routine creates a copy of the path. In block **1001**, if the binding of the passed path entry is simplex (i.e., only one path can be bound to this binding), then the routine continues at block **1002**, else the routine continues at block **1003**. In block **1002**, the routine sets the local pointer path address to point to an address structure that is a copy of the address structure associated with the passed path entry structure with its current binding to the address entry associated with the passed path entry structure, and then returns. In block **1003**, the routine sets the local pointer path address to point to an address structure that contains the URL of the path that contains the passed path entry. In block **1004**, the routine sets the local pointer element to null to initialize the selection of the bindings. In blocks **1005** through **1007**, the routine loops adding all the bindings for the address of the passed path entry that include and are before the passed path entry to the address pointed to by the local path address. In block **1005**, the routine retrieves the next binding from the binding list starting with the first. If there is no such binding, then the routine returns, else the routine continues at block **1006**. In block **1006**, the routine adds the binding to the binding list of the local path address structure and sets the current binding of the local variable path address. In the block **1007**, if the local pointer element is equal to the address entry of the passed path entry, then the routine returns, else the routine loops to block **1005** to select the next binding.

FIG. **11** is a flow diagram of a routine to get the next binding. This routine returns the next binding from the local binding list. If there is no next binding, then the routine invokes the routine label map get to identify the list of edges ("trails") that will map the output label to the target label. If only one trail is identified, then the binding list of path address is extended by the edges of the trail. If multiple trails are identified, then a path is created for each trail and the routine returns so that the demux process can be invoked for each created path. In block **1101**, the routine sets the local pointer binding to point to the next or previous (as indicated by the traverse function) binding in the local binding list. In block **1102**, if a binding was found, then the routine returns an indication that a binding was found, else the routine continues at block **1103**. In block **1103**, the routine invokes the label map get function passing the output label and target label of the local map structure. The label map get function returns a trail list. A trail is a list of edges from the output label to the target label. In decision block **1104**, if the size of the trail list is one, then the routine continues at block **1105**, else the routine continues at block **1112**. In blocks **1105-1111**, the routine extends the binding list by adding a binding data structure for each edge in the trail. The routine then sets the local binding to the last binding in the binding list. In block **1108**, the routine sets the local pointer current binding to point to the last binding in the local binding list. In block

1106, the routine sets the local variable temp trail to the trail in the trail list. In block **1107**, the routine extends the binding list by temp trail by adding a binding for each edge in the trail. These bindings are not yet nailed. In block **1108**, the routine sets the local binding to point to the last binding in the local binding list. In decision block **1109**, if the local binding does not have a key for a session and the local map has a target key for a session, then the routine sets the key for the binding to the target key of the local map and continues at block **1110**, else the routine loops to block **1101** to retrieve the next binding in path. In block **1110**, the routine sets the key of the local binding to the target key of the local map. In block **1111**, the routine sets the target key of the local map to null and then loop to block **1101** to return the next binding. In decision block **1112**, if the local session is set, then the demultiplexing is already in progress and the routine returns a break status. In block **1113**, the routine invokes a prepare multicast paths routine to prepare a path entry for each trail in the trail list. The routine then returns a multiple status.

FIG. **12** is a flow diagram of the get key routine. The get key routine invokes an edge's demuxkey routine to retrieve a key for the session associated with the message. The key identifies the session of a protocol. The demux key routine creates the appropriate key for the message. The demux key routine returns a status of remove, postpone, or other. The status of remove indicates that the current binding should be removed from the path. The status of postpone indicates that the demux key routine cannot create the key because it needs information provided by subsequent protocols in the path. For example, a TCP session is defined by a combination of a remote and local port address and an IP address. Thus, the TCP protocol postpones the creating of a key until the IP protocol identifies the IP address. The get key routine returns a next binding status to continue at the next binding in the path. Otherwise, the routine returns a continue status. In block **1201**, the routine sets the local edge to the edge of the local binding (current binding) and sets the local protocol to the protocol of the local edge. In block **1202**, the routine invokes the demux key routine of the local edge passing the local message, local path address, and local map. The demux key routine sets the key in the local binding. In decision block **1203**, if the demux key routine returns a status of remove, then the routine continues at block **1204**. If the demux key routine returns a status of postpone, then the routine continues at block **1205**, else the routine continues at block **1206**. In block **1204**, the routine sets the flag of the local binding to indicate that the binding is to be removed and continues at block **1206**. In block **1205**, the routine sets the variable traverse to the function to list the next data, increments the variable postpone, and then returns a next binding status. In blocks **1206-1214**, the routine processes the postponing of the creating of a key. In blocks **1207-1210**, if the creating of a key has been postponed, then the routine indicates to backtrack on the path, save the demux status, and set the demux status to demux continue. In blocks **1211-1213**, if the creating of a key has not been postponed, then the routine indicates to continue forward in the path and to restore any saved demux status. The save demux status is the status associated by the binding where the backtrack started. In decision block **1206**, if the variable postpone is set, then the routine continues at block **1207**, else the routine continues at block **1211**. In block **1207**, the routine decrements the variable postpone and sets the variable traverse to the list previous data function. In decision block **1208**, if the variable saved status is set, then the routine continues at block **1210**, else the routine continues at block **1209**. The variable saved status contains the status of the demux process when the demux process started to backtrack.

In block **1209**, the routine sets the variable saved status to the variable status. In block **1210**, the routine sets the variable status to demux continue and continues at block **1214**. In block **1211**, the routine sets the variable traverse to the list next data function. In decision block **1212**, if the variable saved status in set, then the routine continues at block **1213**, else the routine continues at block **1214**. In block **1213**, the routine sets the variable status to the variable saved status and sets the variable saved status to 0. In decision block **1214**, if the local binding indicates that it is to be removed, then the routine returns a next binding status, else the routine returns a continue status.

FIG. **13** is a flow diagram of the get session routine. This routine retrieves the session data structure, creating a data structure session if necessary, for the key indicated by the binding. In block **1301**, the routine retrieves the session from the session table of the local protocol indicated by the key of the local binding. Each protocol maintains a mapping from each key to the session associated with the key. In decision block **1302**, if there is no session, then the routine continues at block **1303**, else the routine returns. In block **1303**, the routine creates a session for the local protocol. In block **1304**, the routine initializes the key for the local session based on the key of the local binding. In block **1305**, the routine puts the session into the session table of the local protocol. In block **1306**, the routine invokes the create session function of the protocol to allow the protocol to initialize its context and then returns.

FIG. **14** is a flow diagram of the nail binding routine. This routine determines whether a binding is already associated with ("nailed to") the session. If so, the routine returns that binding. If not, the routine associates the binding with the session. The routine returns a status of simplex to indicate that only one path can extend through the nailed binding. In decision block **1401**, if the binding table of the session contains an entry for the edge, then the routine continues at block **1402**, else the routine continues at block **1405**. In block **1402**, the routine sets the binding to the entry from the binding table of the local session for the edge. In block **1403**, the routine sets the current binding to point to the binding from the session. In block **1404**, if the binding is simplex, then the routine returns a simplex status, else the routine returns. Blocks **1405** through **1410** are performed when there is no binding in the session for the edge. In block **1405**, the routine sets the session of the binding to the variable session. In block **1406**, the routine sets the key of the binding to the key from the session. In block **1407**, the routine sets the entry for the edge in the binding table of the local session to the binding. In block **1408**, the routine invokes the create binding function of the edge of the binding passing the binding so the edge can initialize the binding. If that function returns a status of remove, the routine continues at block **1409**. In block **1409**, the routine sets the binding to be removed and then returns.

FIG. **15** is a flow diagram of the find path routine. The find path routine identifies the shortest path through the binding list. If no such path exists, then the routine extends a path to include the binding list. In decision block **1501**, if the binding is simplex and a path already goes through this binding (returned as an entry), then the routine continues at block **1502**, else the routine continues at block **1503**. In block **1502**, the routine sets the path to the path of the entry and returns. In block **1503**, the routine initializes the pointers element and short entry to null. In block **1504**, the routine sets the path to the path of the passed path entry. If the local path is not null and its status is demux extend, then the routine continues at block **1509**, else the routine continues at block **1505**. In blocks **1505-1508**, the routine loops identifying the shortest path through the bindings in the binding list. The routine loops selecting each path through the binding. The selected path is eligible if it starts at the first binding in the binding list and the path ends at the binding. The routine loops setting the short entry to the shortest eligible path found so far. In block **1505**, the routine sets the variable first binding to the first binding in the binding list of the path address. In block **1506**, the routine selects the next path (entry) in the path list of the binding starting with the first. If a path is selected (indicating that there are more paths in the binding), then the routine continues at block **1507**, else the routine continues at block **1509**. In block **1507**, the routine determines whether the selected path starts at the first binding in the binding list, whether the selected path ends at the last binding in the binding list, and whether the number of path entries in the selected path is less than the number of path entries in the shortest path selected so far. If these conditions are all satisfied, then the routine continues at block **1508**, else the routine loops to block **1506** to select the next path (entry). In block **1508**, the routine sets the shortest path (short entry) to the selected path and loops to block **1506** to select the next path through the binding. In block **1509**, the routine sets the selected path (entry) to the shortest path. In decision block **1510**, if a path has been found, then the routine continues at block **1511**, else the routine continues at block **1512**. In block **1511**, the routine sets the path to the path of the selected path entry and returns. Blocks **1512-1516** are performed when no paths have been found. In block **1512**, the routine sets the path to the path of the passed path entry. If the passed path entry has a path and its status is demux extend, then the routine continues at block **1515**, else the routine continues at block **1513**. In block **1513**, the routine creates a path for the path address. In block **1514**, the routine sets the variable element to null and sets the path entry to the first element in the stack list of the path. In block **1515**, the routine sets the variable element to be address entry of the member of the passed path entry and sets the path entry to the passed path entry. In block **1516**, the routine invokes the extend path routine to extend the path and then returns. The extend path routine creates a path through the bindings of the binding list and sets the path status to the current demux status.

FIG. **16** is a flow diagram of the process of path hopping routine. Path hopping occurs when the path through the binding list is not the same path as that of the passed path entry. In decision block **1601**, if the path of the passed path entry is set, then the routine continues at block **1602**, else the routine continues at block **1609**. In decision block **1602**, if the path of the passed path entry is equal to the local path, then the routine continues at **1612**, else path hopping is occurring and the routine continues at block **1603**. In blocks **1603-1607**, the routine loops positioning pointers at the first path entries of the paths that are not at the same binding. In block **1603**, the routine sets the variable old stack to the stack list of the path of the passed path entry. In block **1604**, the routine sets the variable new stack to the stack list of the local path. In block **1605**, the routine sets the variable old element to the next element in the old stack. In block **1606**, the routine sets the variable element to the next element in the new stack. In decision block **1607**, the routine loops until the path entry that is not in the same binding is located. In decision block **1608**, if the variable old entry is set, then the routine is not at the end of the hopped from path and the routine continues at block **1609**, else routine continues at block **1612**. In block **1609**, the routine sets the variable entry to the previous entry in the hopped-to path. In block **1610**, the routine sets the path of the passed path entry to the local path. In block **1611**, the routine sets the local entry to the first path entry of the stack list of the local path. In block **1612**, the routine inserts an entry into return list and then returns.

Although the conversion system has been described in terms of various embodiments, the invention is not limited to these embodiments. Modification within the spirit of the invention will be apparent to those skilled in the art. For example, a conversion routine may be used for routing a message and may perform no conversion of the message. Also, a reference to a single copy of the message can be passed to each conversion routine or demuxkey routine. These routines can advance the reference past the header information for the protocol so that the reference is positioned at the next header. After the demux process, the reference can be reset to point to the first header for processing by the conversion routines in sequence. The scope of the invention is defined by the claims that follow.

What is claimed is:

1. An apparatus, comprising:

a processing unit; and

a memory storing instructions executable by the processing unit to:

identify a path for one or more received packets of a message, wherein the path indicates a sequence of two or more routines for processing packets in the message, wherein the path is identified based on a key located in one of the received packets, and wherein the key includes an IP address and a port address; and

process the one or more received packets using the sequence of routines indicated in the identified path, wherein the sequence includes a routine that is used to execute a Transmission Control Protocol (TCP) to convert one or more packets having a TCP format into a different format.

2. The apparatus of claim **1**, wherein the key includes a remote port address and a local port address.

3. The apparatus of claim **1**, wherein the sequence of routines includes:

a second routine that is used to execute a second, different protocol to convert packets of the different format into another format, wherein the second protocol is an application layer protocol.

4. The apparatus of claim **3**, wherein the sequence of routines further includes a third routine that is used to execute a different application layer protocol to further convert the packets.

5. The apparatus of claim **1**, wherein the path further indicates sessions corresponding to respective ones of the sequence of routines.

6. The apparatus of claim **1**, wherein the key identifies a TCP session associated with the received one or more packets.

7. The apparatus of claim **1**, wherein the sequence of routines includes a routine that is executable to process the one or more packets without converting a format of the packets.

8. An apparatus, comprising:

a processing unit; and

a memory storing instructions executable by the processing unit to:

receive one or more packets of a message;

identify, using an IP address and one or more port addresses located in one of the received packets, a sequence of two or more routines for processing packets in the message; and

process the one or more received packets using the identified sequence of routines, wherein the sequence includes a routine that is executable to perform a

Transmission Control Protocol (TCP) to convert at least one of the packets of the message into a different format.

9. The apparatus of claim **8**, wherein the one or more port addresses include a remote port address and a local port address.

10. The apparatus of claim **8**, wherein the sequence of routines includes a plurality of application-level routines.

11. The apparatus of claim **8**, wherein the IP address and the one or more port addresses located in one of the received packets forms a key value that identifies a TCP session associated with the one or more received packets.

12. The apparatus of claim **8**, wherein the instructions are executable to use the IP address and the one or more port addresses to identify sessions corresponding to various ones of the sequence of routines.

13. The apparatus of claim **8**, wherein the instructions are executable to use the IP address and the one or more port addresses to identify a corresponding queue for the message.

14. The apparatus of claim **8**, wherein the sequence of routines includes a routine that does not perform a format conversion on the one or more received packets.

15. A non-transitory, computer-readable medium comprising software instructions for processing a message, wherein the software instructions, when executed, cause a computer system to:

identify a path for one or more received packets of the message, wherein the path indicates a sequence of two or more routines for processing packets in the message,

wherein the path is identified based on a key value located in one of the received packets, and wherein the key value includes an IP address and one or more port addresses;

process the one or more received packets using the sequence of routines indicated in the identified path, wherein the sequence includes a routine that is used to execute a Transmission Control Protocol (TCP) to convert one or more packets having a TCP format into a different format.

16. The computer-readable medium of claim **15**, wherein the one or more port addresses in the key value include a remote port address and a local port address.

17. The computer-readable medium of claim **15**, wherein the path indicates sessions corresponding to respective ones of the sequence of routines.

18. The computer-readable medium of claim **15**, wherein the sequence of routines includes a plurality of application-level routines.

19. The computer-readable medium of claim **18**, wherein the plurality of application-level routines includes a decryption routine.

20. The computer-readable medium of claim **15**, wherein the sequence of routines includes a routine that is used to execute an Internet Protocol (IP) to convert packets having an IP format into the TCP format, and wherein the key value further identifies a TCP session associated with the one or more received packets.

* * * * *

EXHIBIT 6

US009591104B2

(12) United States Patent
Balassanian

(10) **Patent No.: US 9,591,104 B2**
(45) **Date of Patent: *Mar. 7, 2017**

(54) **METHOD AND SYSTEM FOR DATA DEMULTIPLEXING**

(71) Applicant: **Implicit, LLC**, Seattle, WA (US)

(72) Inventor: **Edward Balassanian**, Seattle, WA (US)

(73) Assignee: **Implicit, LLC**, Seattle, WA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 100 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **15/050,027**

(22) Filed: **Feb. 22, 2016**

(65) **Prior Publication Data**

US 2016/0173653 A1 Jun. 16, 2016

Related U.S. Application Data

(63) Continuation of application No. 14/230,952, filed on Mar. 31, 2014, now Pat. No. 9,270,790, which is a continuation of application No. 13/911,324, filed on Jun. 6, 2013, now Pat. No. 8,694,683, which is a continuation of application No. 13/236,090, filed on (Continued)

(51) **Int. Cl.**

H04L 12/58	(2006.01)
H04L 29/06	(2006.01)
H04L 29/08	(2006.01)
H04L 29/12	(2006.01)

(52) **U.S. Cl.**
CPC ***H04L 69/08*** (2013.01); ***H04L 61/2007*** (2013.01); ***H04L 61/6063*** (2013.01); ***H04L 67/02*** (2013.01); ***H04L 69/16*** (2013.01); ***H04L 69/18*** (2013.01)

(58) **Field of Classification Search**
None
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,298,674	A	3/1994	Yun
5,392,390	A	2/1995	Crozier

(Continued)

FOREIGN PATENT DOCUMENTS

EP	0408132	1/1991
EP	0807347	11/1997
EP	0817031	1/1998

OTHER PUBLICATIONS

2015 WL 2194627, United States District Court, N.D. California, *Implicit L.L.C.*, Plaintiff, v. *F5 Networks, Inc.*, Defendant, Case No. 14-cv-02856-SI, signed May 6, 2015, 14 pages.

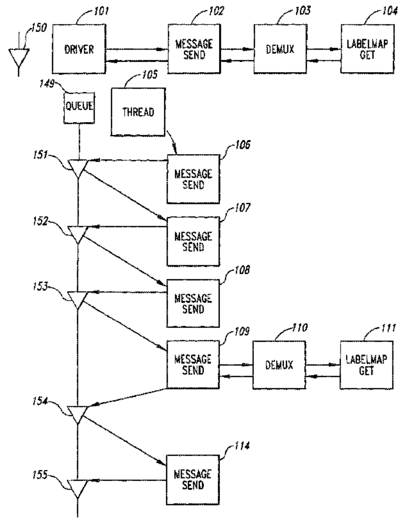
(Continued)

Primary Examiner — Duc Duong

(57) **ABSTRACT**

A method and system for demultiplexing packets of a message is provided.
The demultiplexing system receives packets of a message, identifies a sequence of message handlers for processing the message, identifies state information associated with the message for each message handler, and invokes the message handlers passing the message and the associated state information. The system identifies the message handlers based on the initial data type of the message and a target data type. The identified message handlers effect the conversion of the data to the target data type through various intermediate data types.

20 Claims, 16 Drawing Sheets



Related U.S. Application Data

Sep. 19, 2011, now abandoned, which is a continuation of application No. 10/636,314, filed on Aug. 6, 2003, now Pat. No. 8,055,786, which is a continuation of application No. 09/474,664, filed on Dec. 29, 1999, now Pat. No. 6,629,163.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,414,833	A	5/1995	Hershey et al.	
5,425,029	A	6/1995	Hluchyj et al.	
5,568,478	A	10/1996	van Loo, Jr. et al.	
5,627,997	A	5/1997	Pearson et al.	
5,710,917	A	1/1998	Musa et al.	
5,761,651	A	6/1998	Hasebe et al.	
5,768,521	A	6/1998	Dedrick	
5,826,027	A	10/1998	Pedersen et al.	
5,835,726	A	11/1998	Shwed et al.	
5,848,233	A	12/1998	Radia et al.	
5,848,415	A	12/1998	Guck	
5,854,899	A	12/1998	Callon et al.	
5,870,479	A	2/1999	Feiken et al.	
5,898,830	A	4/1999	Wesinger, Jr. et al.	
5,991,299	A *	11/1999	Radogna	H04L 29/06 370/389
5,991,806	A *	11/1999	McHann, Jr.	H04L 41/0213 709/224
6,047,002	A	4/2000	Hartmann et al.	
6,091,725	A	7/2000	Cheriton et al.	
6,101,189	A	8/2000	Tsuruoka	
6,101,320	A	8/2000	Schuetze et al.	
6,104,500	A	8/2000	Alam et al.	
6,104,704	A	8/2000	Buhler et al.	
6,111,893	A	8/2000	Volftsun et al.	
6,115,393	A	9/2000	Engel et al.	
6,119,236	A	9/2000	Shipley	
6,128,624	A	10/2000	Papierniak et al.	
6,141,749	A	10/2000	Coss et al.	
6,151,390	A	11/2000	Volftsun et al.	
6,157,622	A	12/2000	Tanaka et al.	
6,192,419	B1	2/2001	Aditham et al.	
6,199,054	B1	3/2001	Khan et al.	
6,212,550	B1	4/2001	Segur	
6,222,536	B1	4/2001	Kihl et al.	
6,226,267	B1	5/2001	Spinney et al.	
6,243,667	B1	6/2001	Kerr et al.	
6,246,678	B1	6/2001	Erb et al.	
6,259,781	B1	7/2001	Crouch et al.	
6,275,507	B1	8/2001	Anderson et al.	
6,278,532	B1	8/2001	Heimendinger et al.	
6,356,529	B1	3/2002	Zarom	
6,359,911	B1	3/2002	Movshovich et al.	
6,401,132	B1	6/2002	Bellwood et al.	
6,404,775	B1	6/2002	Leslie et al.	
6,405,254	B1	6/2002	Hadland	
6,426,943	B1	7/2002	Spinney et al.	
6,493,348	B1	12/2002	Gelman et al.	
6,504,843	B1	1/2003	Cremin et al.	
6,519,636	B2	2/2003	Engel et al.	
6,560,236	B1	5/2003	Varghese et al.	
6,574,610	B1	6/2003	Clayton et al.	
6,598,034	B1	7/2003	Kloth	
6,629,163	B1	9/2003	Balassanian	
6,651,099	B1	11/2003	Dietz et al.	
6,678,518	B2	1/2004	Eerola	
6,680,922	B1	1/2004	Jorgensen	
6,701,432	B1	3/2004	Deng et al.	
6,711,166	B1	3/2004	Amir et al.	
6,785,730	B1	8/2004	Taylor	
6,871,179	B1	3/2005	Kist et al.	
6,889,181	B2	5/2005	Kerr et al.	
6,937,574	B1	8/2005	Delaney et al.	
6,957,346	B1	10/2005	Kivinen et al.	
6,959,439	B1 *	10/2005	Boike	G06F 13/102 719/321
7,233,569	B1	6/2007	Swallow	
7,233,948	B1	6/2007	Shamoon et al.	
7,281,036	B1	10/2007	Lu et al.	
7,383,341	B1	6/2008	Saito et al.	
7,711,857	B2	5/2010	Balassanian	
8,055,786	B2	11/2011	Balassanian	
8,694,683	B2	4/2014	Balassanian	
2003/0142669	A1	7/2003	Kubota et al.	
2004/0015609	A1 *	1/2004	Brown	H04L 69/08 709/246
2008/0250045	A1	10/2008	Balassanian et al.	
2009/0083763	A1 *	3/2009	Sareen	H04L 67/2823 719/317
2009/0265695	A1	10/2009	Karino	

OTHER PUBLICATIONS

Alexander, D. et al., "The SwitchWare Active Network Architecture", Jun. 6, 1998, IEEE.
Antoniazzi, S. et al., "An Open Software Architecture for Multimedia Consumer Terminals", Central Research Labs, Italy; Alcatel SEL Research Centre, Germany, ECMAST 1997.
Arbanowski, Stefan, "Generic Description of Telecommunication Services and Dynamic Resource Selection in Intelligent Communication Environments", Thesis, Technische Universitat Berlin, Oct. 9, 1996 (3 documents).
Arbanowski, S., et al.,Service Personalization for Unified Messaging Systems, Jul. 6-8, 1999, The Fourth IEEE Symposium on Computers and Communications, ISCC '99, Red Sea, Egypt.
Atkinson, R., "Security Architecture for the Internet Protocol", Aug. 1995, Naval Research Laboratory.
Atkinson, R., "IP Authentication Header", Aug. 1995, Naval Research Laboratory.
Atkinson, R., "IP Encapsulating Security Payload (ESP)", Aug. 1995, Naval Research Laboratory.
Back, G., et al., Java Operating Systems: Design and Implementation, Aug. 1998, Technical Report UUCS-98-015, University of Utah.
Baker, Dr. Sean, "CORBA Implementation Issues", 1994, IONA Technologies, O'Reilly Institute Dublin, Ireland.
Barrett, R., et al., "Intermediaries: New Places for Producing and Manipulating Web Content", 1998, IBM Almaden Research Center, Elsevier Science.
Bellare, M., et al., "A Concrete Security Treatment of Symmetric Encryption: Analysis of the DES Modes of Operation", Aug. 15, 1997, Dept. of Computer Science and Engineering, University of California, San Diego.
Bellare, M., et al., "A Concrete Security Treatment of Symmetric Encryption: Analysis of the DES Modes of Operation", Aug. 15, 1997, IEEE.
Bellare, M., et al., "XOR MACs: New Methods for Message Authentication Using Finite Pseudorandom Functions", 1995, CRYPTO '95, LNCS 963, pp. 15-28, Springer-Verlag Berlin Heidelberg.
Bellissard, L., et al., "Dynamic Reconfiguration of Agent-Based Applications", Third European Research Seminar on Advances in Distributed Systems, (ERSADS '99) Madeira Island.
Bolding, Darren, "Network Security, Filters and Firewalls", 1995, www.acm.org/crossroads/xrds2-1/security.html.
Booch, G., et al., "Software Engineering with ADA", 1994, Third Edition, The Benjamin/Cummings Publishing Company, Inc. (2 documents).
Breugst, et al., "Mobile Agents—Enabling Technology for Active Intelligent Network Implementation", May/Jun. 1998, IEEE Network.
"C Library Functions", AUTH(3) Sep. 17, 1993, Solbourne Computer, Inc.
Chapman, D., et al., "Building Internet Firewalls", Sep. 1995, O'Reilly & Associates, Inc.
CheckPoint FireWall-1 Technical White Paper, Jul. 18, 1994, CheckPoint Software Technologies, Ltd.
CheckPoint FireWall-1 White Paper, Sep. 1995, Version 2.0, CheckPoint Software Technologies, Ltd.

(56) **References Cited**

OTHER PUBLICATIONS

Command Line Interface Guide P/N 093-0011-000 Rev C Version 2.5, 2000-2001, NetScreen Technologies, Inc.
Coulson, G. et al., "A CORBA Compliant Real-Time Multimedia Platform for Broadband Networks", Lecture Notes in Computer Science, 1996, Trends in Distributed Systems CORBA and Beyond.
Cox, Brad, "SuperDistribution, Objects As Property on the Electronic Frontier", 1996, Addison-Wesley Publishing Company.
Cranes, et al., "A Configurable Protocol Architecture for CORBA Environments", Autonomous Decentralized Systems 1997 Proceedings ISADS, Third International Symposium Apr. 9-11, 1997.
Curran, K., et al., "CORBA Lacks Venom", University of Ulster, Northern Ireland, UK 2000.
Dannert, Andreas, "Call Logic Service for a Personal Communication Supporting System", Thesis, Jan. 20, 1998, Technische Universitat Berlin.
DARPA Internet Program Protocol Specification, "Transmission Control Protocol", Sep. 1981, Information Sciences Institute, California.
DARPA Internet Program Protocol Specification, "Internet Protocol", Sep. 1981, Information Sciences Institute, California.
Decasper, D., et al., "Crossbow: A Toolkit for Integrated Services over Cell Switched IPv6", 1997, Computer Engineering and Networks Laboratory, ETH Zurich, Switzerland.
Decasper, D., et al., "Router Plugins A Software Architecture for Next Generation Routers", 1998, Proceedings of ACM SIGCONM '98.
Deering, S., et al., Internet Protocol, Version 6 (IPv6) Specification, Dec. 1998, Nokia, The Internet Society.
Deering, S., et al., Internet Protocol, Version 6 (IPv6) Specification, Dec. 1995, Network Working Group, RFC 1883.
Dutton, et al, "Asynchronous Transfer Mode Technical Overview (ATM)", Second Edition; IBM, Oct. 1995, 2nd Edition, Prentice Hall PTR, USA.
Eckardt, T., et al., "Application of X.500 and X.700 Standards for Supporting Personal Communications in Distributed Computing Environments", 1995, IEEE.
Eckardt, T., et al., "Personal Communications Support based on TMN and TINA Concepts", 1996, IEEE Intelligent Network Workshop (IN '96), Apr. 21-24, Melbourne, Australia.
Eckardt, T., et al., "Beyond IN and UPT—A Personal Communications Support System Based on TMN Concepts", Sep. 1997, IEEE Journal on Selected Areas in Communications, vol. 15, No. 7.
Egevang, K., et al., "The IP Network Address Translator (NAT)", May 1994, Network Working Group, RFC 1631.
Estrin, D., et al., "Visa Protocols for Controlling Inter-Organizational Datagram Flow", Dec. 1998, Computer Science Department, University of Southern California and Digital Equipment Corporation.
Faupel, M., "Java Distribution and Deployment", Oct. 9, 1997, APM Ltd., United Kingdom.
Felber, P., "The CORBA Object Group Service: A Service Approach to Object Groups in CORBA", Thesis, 1998, Ecole Polytechnique Federate de Lausanne, Switzerland.
Fish, R., et al., "DRoPS: Kernel Support for Runtime Adaptable Protocols", Aug. 25-27, 1998, IEEE 24th Euromicro Conference, Sweden.
Fiuczynski, M., et al., "An Extensible Protocol Architecture for Application-Specific Networking", 1996, Department of Computer Science and Engineering, University of Washington.
Franz, Stefan, "Job and Stream Control in Heterogeneous Hardware and Software Architectures", Apr. 1998, Technische Universitat, Berlin (2 documents).
Fraser, T., "DTE Firewalls: Phase Two Measurement and Evaluation Report", Jul. 22, 1997, Trusted Information Systems, USA.
Gazis, V., et al., "A Survey of Dynamically Adaptable Protocol Stacks", first Quarter 2010, IEEE Communications Surveys & Tutorials, vol. 12, No. 1, 1st Quarter.
Gokhale, A., et al., "Evaluating the Performance of Demultiplexing Strategies for Real-Time CORBA", Nov. 1997, GLOBECOM.
Gokhale, A., et al., "Measuring and Optimizing CORBA Latency and Scalability Over High-Speed Networks", Apr. 1998, IEEE Transaction on Computers, vol. 47, No. 4; Proceedings of the International Conference on Distributed Computing Systems (ICDCS '97) May 27-30, 1997.
Gokhale, A., et al., "Operating System Support for High-Performance, Real-Time CORBA", 1996.
Gokhale, A., et al., "Principles for Optimizing CORBA Internet Inter-ORB Protocol Performance", Jan. 9, 1998, Proceedings of the HICSS Conference, Hawaii.
Gong, Li, "Java Security: Present and Near Future", May/Jun. 1997, IEEE Micro.
Gong, Li, "New Security Architectural Directions for Java (Extended Abstract)", Dec. 19, 1996, IEEE.
Gong, Li, "Secure Java Class Loading", Nov./Dec. 1998, IEEE Internet.
Goos, G., et al., "Lecture Notes in Computer Science: Mobile Agents and Security", 1998, Springer-Verlag Berlin Heidelberg.
Goralski, W., "Introduction to ATM Networking", 1995, McGraw-Hill Series on Computer Communications, USA.
Hamzeh, K., et al., "Layer Two Tunneling Protocol L2TP", Jan. 1998, PPP Working Group, Internet Draft.
Harrison, T., et al., "The Design and Performance of a Real-Time CORBA Event Service", Aug. 8, 1997,Proceedings of the OOPSLA '97 Conference, Atlanta, Georgia in Oct. 1997.
Huitema, Christian, "IPv6 The New Internet Protocol", 1997 Prentice Hall, Second Edition.
Hutchins, J., et al., "Enhanced Internet Firewall Design Using Stateful Filters Final Report", Aug. 1997, Sandia Report; Sandia National Laboratories.
IBM, Local Area Network Concepts and Products: Routers and Gateways, May 1996.
Juniper Networks Press Release, Juniper Networks Announces Junos, First Routing Operating System for High-Growth Internet Backbone Networks, Jul. 1, 1998, Juniper Networks.
Juniper Networks Press Release, Juniper Networks Ships the Industry's First Internet Backbone Router Delivering Unrivaled Scalability, Control and Performance, Sep. 16, 1998, Juniper Networks.
Karn, P., et al., "The ESP DES-CBC Transform", Aug. 1995, Network Working Group, RFC 1829.
Kelsey, J. et al., "Authenticating Outputs of Computer Software Using a Cryptographic Coprocessor", Sep. 1996, CARDIS.
Krieger, D., et al., "The Emergence of Distributed Component Platforms", Mar. 1998, IEEE.
Krupczak, B., et al., "Implementing Communication Protocols in Java", Oct. 1998, IEEE Communications Magazine.
Krupczak, B., et al., "Implementing Protocols in Java: The Price of Portability", 1998, IEEE.
Lawson, Stephen, "Cisco NetFlow Switching Speeds Traffic Routing", Jul. 7, 1997, Infoworld.
Li, S., et al., "Active Gateway: A Facility for Video Conferencing Traffic Control", Feb. 1, 1997, Purdue University; Purdue e-Pubs; Computer Science Technical Reports.
Magedanz, T., et al., "Intelligent Agents: An Emerging Technology for Next Generation Telecommunications?", 1996, IEEE.
Mills, H., et al., "Principles of Information Systems Analysis and Design", 1986, Academic Press, Inc. (2 documents).
Mosberger, David, "*Scout: A Path-Based Operating System*", Doctoral Dissertation Submitted to the University of Arizona, 1997 (3 documents).
Muhugusa, M., et al., "COMSCRIPT : An Environment for the Implementation of Protocol Stacks and their Dynamic Reconfiguration", Dec. 1994.
Nelson, M., et al., The Data Compression Book, 2nd Edition, 1996, M&T Books, A division of MIS Press, Inc.
NetRanger User's Guide, 1996, WheelGroup Corporation.
NetScreen Command Line Reference Guide, 2000, P/N 093-0000-001 Rev A, NetScreen Technologies, Inc., USA.
NetScreen Command Line Reference Guide, 2000, P/N 093-0000-001 NetScreen Technologies, Inc., USA.
NetScreen Concepts and Examples ScreenOS Reference Guide, 1998-2001, Version 2.5 P/N 093-0039-000 Rev. A, NetScreen Technologies, Inc.

(56) **References Cited**

OTHER PUBLICATIONS

NetScreen Products Webpage, wysiwyg://body_bottom.3/http://www...een.com/products/products.html 1998-1999, NetScreen Technologies, Inc.
NetScreen WebUI, Reference Guide, Version 2.5.0 P/N 093-0040-000 Rev. A, 2000-2001, NetScreen Technologies, Inc.
NetStalker Installation and User's Guide, 1996, Version 1.0.2, Haystack Labs, Inc.
Niculescu, Dragos, "Survey of Active Network Research", Jul. 14, 1999, Rutgers University.
Nortel Northern Telecom, "ISDN Primary Rate User-Network Interface Specification", Aug. 1998.
Nygren, Erik, "The Design and Implementation of a High-Performance Active Network Node", Thesis, Feb. 1998, MIT.
Osbourne, E., "Morningstar Technologies SecureConnect Dynamic Firewall Filter User's Guide", Jun. 14, 1995, V. 1.4, Morning Star Technologies, Inc.
Padovano, Michael, "Networking Applications on UNIX System V Release 4," 1993 Prentice Hall, USA (2 documents).
Pfeifer, T., "Automatic Conversion of Communication Media", 2000, GMD Research Series, Germany.
Pfeifer, T., "Automatic Conversion of Communication Media", Thesis, 1999, Technischen Universitat Berlin, Berlin.
Pfeifer, T., et al., "Applying Quality-of-Service Parametrization for Medium-to-Medium Conversion", Aug. 25-28, 1996, 8th IEEE Workshop on Local and Metropolitan Area Networks, Potsdam, Germany.
Pfeifer, T., "Micronet Machines—New Architectural Approaches for Multimedia End-Systems", 1993 Technical University of Berlin.
Pfeifer, T., "On the Convergence of Distributed Computing and Telecommunications in the Field of Personal Communications", 1995, KiVS, Berlin.
Pfeifer, T., "Speech Synthesis in the Intelligent Personal Communication Support System (IPCSS)", Nov. 2-3, 1995, 2nd 'Speak!' Workshop on Speech Generation in Multimodal Information Systems and Practical Applications.
Pfeifer, T., et al., "Generic Conversion of Communication Media for Supporting Personal Mobility", Nov. 25-27, 1996, Proc. of the Third COST 237 Workshop: Multimedia Telecommunications and Applications.
Pfeifer, T., et al., "Intelligent Handling of Communication Media", Oct. 29-31, 1997, 6th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS) Tunis.
Pfeifer, T., et al., "Resource Selection in Heterogeneous Communication Environments using the Teleservice Descriptor", Dec. 15-19, 1997, Proceedings from the 4th COST 237 Workshop: From Multimedia Services to Network Services, Lisboa.
Pfeifer, T., et al., Mobile Guide—Location-Aware Applications from the Lab to the Market, 1998, IDMS '98, LNCS 1483, pp. 15-28.
Pfeifer, T., et al., "The Active Store providing Quality Enhanced Unified Messaging", Oct. 20-22, 1998, 5th Conference on computer Communications, AFRICOM-CCDC '98, Tunis.
Pfeifer, T.,, et al., "*A Modular Location-Aware Service and Application Platform*", 1999, Technical University of Berlin.
Plagemann, T., et al., "*Evaluating Crucial Performance Issues of Protocol Configuration in DaCaPo*", 1994, University of Oslo.
Psounis, Konstantinos, "*Active Networks: Applications, Security Safety, and Architectures*", First Quarter 1999, IEEE Communications Surveys.
Rabiner, Lawrence, "*Applications of Speech Recognition in the Area of Telecommunications*", 1997, IEEE.
Raman, Suchitra, et al, "*A Model, Analysis, and Protocol Framework for Soft State-based Communications*", Department of EECS, University of California, Berkeley.
Rogaway, Phillip, "*Bucket Hashing and its Application to Fast Message Authentication*", Oct. 13, 1997, Department of Computer Science, University of California.
Schneier, B., et al., "*Remote Auditing of Software Outputs Using a Trusted CoProcessor*", 1997, Elsevier Paper Reprint 1999.
Tennenhouse, D., et al., "*From Internet to ActiveNet*", Laboratory of Computer Science, MIT, 1996.
Tudor, P., "*Tutorial MPEG-2 Video Compression*", Dec. 1995, Electronics & Communication Engineering Journal.
US Copyright Webpage of Copyright Title, "*IPv6: the New Internet Protocol*", by Christian Huitema, 1998 Prentice Hall.
Van der Meer, et al., "*An Approach for a 4th Generation Messaging System*", Mar. 21-23, 1999, The Fourth International Symposium on Autonomous Decentralized Systems ISADS '99, Tokyo.
Van der Meer, Sven, "*Dynamic Configuration Management of the Equipment in Distributed Communication Environments*", Thesis, Oct. 6, 1996, Berlin (3 documents).
Van Renesse, R. et al., "*Building Adaptive Systems Using Ensemble*", Cornell University Jul. 1997.
Venkatesan, R., et al., "*Threat-Adaptive Security Policy*", 1997, IEEE.
Wetherall, D., et al., "*The Active IP Option*", Sep. 1996, Proceedings of the 7th ACM SIGOPS European Workshop, Connemara, Ireland.
Welch, Terry, "*A Technique for High-Performance Data Compression*", 1984, Sperry Research Center, IEEE.
Zeletin, R. et al., "*Applying Location Aware Computing for Electronic Commerce: Mobile Guide*", Oct. 20-22, 1998, 5th Conference on Computer Communications, AFRICOM-CCDC '98, Tunis.
Zell, Markus, "*Selection of Converter Chains by Means of Quality of Service Analysis*", Thesis, Feb. 12, 1998, Technische Universitat Berlin.
Feb. 4, 2008 Plaintiff's Original Complaint.
Aug. 26, 2008 Defendant NVIDIA Corporation's Answer to Complaint.
Aug. 26, 2008 Defendant Sun Microsystems, Inc.'s Answer to Complaint.
Aug. 27, 2008 Defendant Advanced Micro Devices, Inc.'s Answer to Complaint for Patent Infringement.
Aug. 27, 2008 RealNetworks, Inc.'s Answer to Implicit Networks, Inc.'s Original Complaint for Patent Infringement, Affirmative Defenses, and Counterclaims.
Aug. 27, 2008 Intel Corp.'s Answer, Defenses and Counterclaims.
Aug. 27, 2008 Defendant RMI Corporation's Answer to Plaintiff's Original Complaint.
Sep. 15, 2008 Plaintiffs Reply to NVIDIA Corporation's Counterclaims.
Sep. 15, 2008 Plaintiff's Reply to Sun Microsystems Inc.'s Counterclaims.
Sep. 16, 2008 Plaintiff's Reply to RealNetworks, Inc.'s Counterclaims.
Sep. 16, 2008 Plaintiff's Reply to Intel Corp.'s Counterclaims.
Dec. 10, 2008 Order granting Stipulated Motion for Dismissal with Prejudice re NVIDIA Corporation, Inc.
Dec. 16, 2008 Defendants AMD, RealNetworks, RMI, and Sun's Motion to Stay Pending the Patent and Trademark Office's Reexamination of the '163 Patent.
Dec. 29, 2008 Order granting Stipulated Motion for Dismissal without Prejudice of Claims re Sun Microsystems, Inc.
Jan. 5, 2009 Plaintiff's Opposition to Defendants AMD, RealNetworks, RMI, and Sun's Motion to Stay Pending Reexamination and Exhibit A.
Jan. 9, 2009 Reply of Defendants AMD, RealNetworks, RMI, and Sun's Motion to Stay Pending the Patent and Trademark Office's Reexamination of the '163 Patent.
Feb. 9, 2009 Order Granting Stay Pending the United States Patent and Trademark Office's Reexamination of U.S. Pat. No. 6,629,163.
Feb. 17, 2009 Order Granting Stipulated Motion for Dismissal of Advanced Micro Devices, Inc. with Prejudice.
May 14, 2009 Order Granting Stipulated Motion for Dismissal of RMI Corporation with Prejudice.
Oct. 13, 2009 Order Granting Stipulated Motion for Dismissal of Claims Against and Counterclaims by Intel Corporation.
Oct. 30, 2009 Executed Order for Stipulated Motion for Dismissal of Claims Against and Counterclaims by RealNetworks, Inc.
Nov. 30, 2009 Plaintiffs Original Complaint, *Implicit* v *Microsoft*, Case No. 09-5628.
Jan. 22, 2010 Order Dismissing Case, *Implicit* v *Microsoft*, Case No. 09-5628.

(56) **References Cited**

OTHER PUBLICATIONS

Aug. 16, 2010 Plaintiffs Original Complaint, *Implicit* v *Cisco*, Case No. 10-3606.
Nov. 22, 2010 Defendant Cisco Systems, Inc.'s Answer and Counterclaims, *Implicit* v *Cisco*, Case No. 10-3606.
Dec. 13, 2010 Plaintiff, Implicit Networks, Inc.'s, Answer to Counterclaims, *Implicit* v *Cisco*, Case No. 10-3606.
Oct. 4, 2011 Order of Dismissal with Prejudice, *Implicit* v *Cisco*, Case No. 10-3606.
Aug. 24, 2010 Plaintiffs Original Complaint, *Implicit* v *Citrix*, Case No. 10-3766.
Dec. 1, 2010 Plaintiff's First Amended Complaint, *Implicit* v *Citrix*, Case No. 10-3766.
Jan. 14, 2011 Defendant Citrix Systems, Inc.'s Answer, Defenses and Counter-complaint for Declaratory Judgment, *Implicit* v *Citrix*, Case No. 10-3766.
Feb. 18, 2011 Plaintiff, Implicit Networks, Inc.'s, Answer to Defendants Counterclaims, *Implicit* v *Citrix*, Case No. 10-3766.
May 2, 2011 Order of Dismissal, *Implicit* v *Citrix*, Case No. 10-3766.
Jul. 30, 2010 Plaintiff's Original Complaint, *Implicit* v *F5*, Case No. 10-3365.
Oct. 13, 2010 Defendants' Answer and Counter-Complaint, *Implicit* v *F5*, Case No. 10-3365.
Nov. 3, 2010 Plaintiff's Answer to Counter-Complaint, *Implicit* v *F5*, Case No. 10-3365.
Dec. 10, 2010 Plaintiff's First Amended Complaint, *Implicit* v *F5*, Case No. 10-3365.
Jan. 14, 2011 Defendants' Answer to 1st Amended Complaint and Counterclaim, *Implicit* v *F5*, Case No. 10-3365.
Feb. 18, 2011 Plaintiff's Answer to F5's Amended Counter-Complaint, *Implicit* v *F5*, Case No. 10-3365.
Apr. 18, 2011 Defendants' Amended Answer to 1st Amended Complaint and Counter-Complaint, *Implicit* v *F5*, Case No. 10-3365.
May 5, 2011 Plaintiff's Answer to F5's Amended Counter-Complaint, *Implicit* v *F5*, Case No. 10-3365.
Jul. 22, 2011 F5 Networks, Inc.'s Invalidity Contentions, *Implicit* v *F5*, Case No. 10-3365.
Jul. 22, 2011 F5 Networks, Inc.'s Invalidity Contentions, Exhibit A, *Implicit* v *F5*, Case No. 10-3365 (31 documents).
Jul. 22, 2011 F5 Networks, Inc.'s Invalidity Contentions, Exhibit B, *Implicit* v *F5*, Case No. 10-3365.
Oct. 18, 2011 Joint Claim Construction & Pre-Hearing Statement (PR 4-3), *Implicit* v *F5*, Case No. 10-3365.
Oct. 18, 2011 Joint Claim Construction & Pre-Hearing Statement (PR 4-3) Exhibit A, *Implicit* v *F5*, Case No. 10-3365 (2 documents).
Nov. 28, 2011 Plaintiff's Opening Claim Construction Brief, *Implicit* v *F5*, Case No. 10-3365.
Nov. 29, 2011 Amended Joint Claim Construction & Pre-Hearing Statement, *Implicit* v *F5*, Case No. 10-3365.
Nov. 29, 2011 Amended Joint Claim Construction & Pre-Hearing Statement, Exhibit A, *Implicit* v *F5*, Case No. 10-3365.
Dec. 12, 2011 Defendants' Claim Construction Brief, *Implicit* v *F5*, Case No. 10-3365.
Dec. 19, 2011 Plaintiff's Reply to Defendants' (F5, HP, Juniper) Responsive Claim Construction Brief (4-5), *Implicit* v *F5*, Case No. 10-3365.
Jan. 27, 2012 Transcript of Proceeding Held on Jan. 17, 2012; *Implicit* v *F5*, Case No. 10-3365.
Jan. 27, 2012 Transcript of Proceeding Held on Jan. 18, 2012; *Implicit* v *F5*, Case No. 10-3365.
Jan. 27, 2012 Transcript of Proceeding Held on Jan. 19, 2012; *Implicit* v *F5*, Case No. 10-3365.
Feb. 29, 2012 Claim Construction Order.
Aug. 15, 2012 Storer Invalidity Report.
Sep. 10, 2012 Implicit's Expert Report of Scott M. Nettles.
Mar. 13, 2013 Order Granting Defendants' Motion for Summary Judgment.
Apr. 9, 2013 Notice of Appeal to the Federal Circuit.
Aug. 23, 2010 Plaintiff's Original Complaint, *Implicit* v *HP*, Case No. 10-3746.
Nov. 23, 2010 Plaintiff's First Amended Complaint, *Implicit* v *HP*, Case No. 10-3746.
Jan. 14, 2011 Defendant HP's Answer and Counterclaims, *Implicit* v *HP*, Case No. 10-3746.
Feb. 18, 2011 Implicit Networks, Inc.'s Answer to HP Counterclaims, *Implicit* v *HP*, Case No. 10-3746.
May 10, 2011 Plaintiff's Amended Disclosure of Asserted Claims and Infringement Contentions, Case No. 10-3746.
Jun. 30, 2011 Defendant HP Company's Invalidity Contentions, *Implicit* v *HP*, Case No. 10-3746.
Jun. 30, 2011 Defendant HP Company's Invalidity Contentions, A1-14, *Implicit* v *HP*, Case No. 10-3746.
Jun. 30, 2011 Defendant HP Company's Invalidity Contentions, B1-21, *Implicit* v *HP*, Case No. 10-3746.
Sep. 20, 2010 Plaintiff's Original Complaint, *Implicit* v *Juniper*, Case No. 10-4234.
Nov. 12, 2010 Juniper Network's Motion to Dismiss For Failure to State a Claim Under Rule 12(B)(6): Memorandum of Points and Authorities; *Implicit* v *Juniper*, Case No. 10-4234.
Nov. 12, 2010 Juniper Network's Request for Judicial Notice in Support of its Motion to Dismiss For Failure to State a Claim Under Rule 12(B)(6): Memorandum of Points and Authorities; *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 1, 2010 First Amended Complaint; *Implicit* v *Juniper*, Case No. 10-4234.
Jan. 18, 2011 Juniper Networks, Inc.'s Answer and Affirmative Defenses to 1st Amended Complaint, *Implicit* v *Juniper*, Case No. 10-4234.
Feb. 18, 2011 Plaintiff's Answer to Defendant's Counterclaims, *Implicit* v *Juniper*, Case No. 10-4234.
May 23, 2011 Plaintiff's Disclosure of Asserted Claims and Infringement Contentions, *Implicit* v *Juniper*, Case No. 10-4234.
Nov. 15, 2011 Plaintiff's Amended Disclosure of Asserted Claim and Infringement Contentions, *Implicit* v *Juniper*, Case No. 10-4234.
Nov. 28, 2011 Spencer Hosie Declaration in Support of Plaintiff's Opening Claim Construction Brief), *Implicit* v *Juniper*, Case No. 10-4234.
Nov. 28, 2011 Spencer Hosie Declaration in Support of Plaintiff's Opening Claim Construction Brief Exhibit E, *Implicit* v *Juniper*, Case No. 10-4234.
Nov. 28, 2011 Spencer Hosie Declaration in Support of Plaintiff's Opening Claim Construction Brief Exhibit J, *Implicit* v *Juniper*, Case No. 10-4234.
Nov. 28, 2011 Spencer Hosie Declaration in Support of Plaintiff's Opening Claim Construction Brief Exhibit K, *Implicit* v *Juniper*, Case No. 10-4234.
Nov. 28, 2011 Spencer Hosie Declaration in Support of Plaintiff's Opening Claim Construction Brief Exhibits M-O, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit B, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit F, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit N, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit P, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit Q, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit S., *Implicit* v *Juniper*, Case No. 10-4234.

(56) **References Cited**

OTHER PUBLICATIONS

Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit T-1, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit T-2, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit T-3, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit T-4, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit U, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit V, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit W, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit X, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit Y-1, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit Y-2, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit Y-3, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit Y-4, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit Z, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 19, 2011 Spencer Hosie Declaration in Support of Plaintiff's Reply Claim Construction Brief, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 19, 2011 Spencer Hosie Declaration in Support of Plaintiff's Reply Claim Construction Brief, Exhibit P, *Implicit* v *Juniper*, Case No. 10-4234.
Jan. 10, 2012 Plaintiff's Jan. 10, 2012 Amended Disclosure of Asserted Claims and Infringement Contentions, *Implicit* v *Juniper*, Case No. 10-4234.
Feb. 10, 2012 Juniper Networks, Inc.'s Supplemental Invalidity Contentions, *Implicit* v *Juniper*, Case No. 10-4234.
Feb. 10, 2012 Juniper Networks, Inc.'s Supplemental Invalidity Contentions, Exhibit A1, *Implicit* v *Juniper*, Case No. 10-4234.
Feb. 10, 2012 Juniper Networks, Inc.'s Supplemental Invalidity Contentions, Exhibit A2, *Implicit* v *Juniper*, Case No. 10-4234.
Feb. 10, 2012 Juniper Networks, Inc.'s Supplemental Invalidity Contentions, Exhibit A3, *Implicit* v *Juniper*, Case No. 10-4234.
Feb. 10, 2012 Juniper Networks, Inc.'s Supplemental Invalidity Contentions, Exhibit A4, *Implicit* v *Juniper*, Case No. 10-4234.
Feb. 10, 2012 Juniper Networks, Inc.'s Supplemental Invalidity Contentions, Exhibit B1, *Implicit* v *Juniper*, Case No. 10-4234.
Feb. 29, 2012 Plaintiff's Feb. 29, 2012 Amended Disclosure of Asserted Claims and Infringement Contentions, *Implicit* v *Juniper*, Case No. 10-4234.
Apr. 6, 2012 Plaintiff's Apr. 6, 2012 Amended Disclosure of Asserted Claims and Infringement Contentions, *Implicit* v *Juniper*, Case No. 10-4234.
Apr. 9, 2012 Plaintiff's Apr. 9, 2012 Amended Disclosure of Asserted Claims and Infringement Contentions, *Implicit* v *Juniper*, Case No. 10-4234.
Sep. 11, 2012 Implicit's Expert Report of Scott Nettles.
Nov. 9, 2012 Juniper's Notice of Motion and Memorandum of Law ISO Motion for Summary Judgment or, in the alternative, for Partial Summary Judgment, on the Issue of Invalidity.
Nov. 9, 2012 Exhibit 2 to Declaration in support of Juniper's Motion for Summary Judgment—Calvert Expert Report.
Nov. 9, 2012 Exhibit 3 to Declaration in support of Juniper's Motion for Summary Judgment—Calvert Supplemental Expert Report.
Nov. 26, 2012 Implicit Opposition to Juniper's and F5 Motion on Invalidity.
Nov. 26, 2012 Exhibit A to Hosie Declaration—Aug. 27, 2012 Excerpts from David Blaine deposition.
Nov. 26, 2012 Exhibit B to Hosie Declaration—Oct. 25, 2012 Excerpts from Kenneth Calvert Deposition.
Nov. 26, 2012 Exhibit C to Hosie Declaration—Aug. 15, 2012 Excerpts from Kenneth Calvert Expert Report.
Nov. 26, 2012 Exhibit D to Hosie Declaration—U.S. Pat. No. 6,651,099 to Dietz et al.
Nov. 26, 2012 Exhibit E to Hosie Declaration—Understanding Packet-Based and Flow-Based Forwarding.
Nov. 26, 2012 Exhibit F to Hosie Declaration—Wikipedia on Soft State.
Nov. 26, 2012 Exhibit G to Hosie Declaration—Sprint Notes.
Nov. 26, 2012 Exhibit H to Hosie Declaration—Implicit's Supplemental Response to Juniper's 2nd Set of Interrogatories.
Nov. 26, 2012 Exhibit I to Hosie Declaration—U.S. Pat. No. 7,650,634 (Zuk).
U.S. Appl. No. 11/933,022 Utility Application filed Oct. 31, 2007.
U.S. Appl. No. 11/933,022 Preliminary Amendment filed Feb. 19, 2008.
U.S. Appl. No. 11/933,022 Office Action mailed Jun. 24, 2009.
U.S. Appl. No. 11/933,022 Amendment filed Sep. 24, 2009.
U.S. Appl. No. 11/933,022 Office Action dated Dec. 11, 2009.
U.S. Appl. No. 11/933,022 Amendment and Response dated Jan. 29, 2010.
U.S. Appl. No. 11/933,022 Notice of Allowance dated Mar. 2, 2010.
U.S. Appl. No. 11/933,022 Issue Notification dated May 4, 2010.
U.S. Appl. No. 10/636,314 Utility Application filed Aug. 6, 2003.
U.S. Appl. No. 10/636,314 Office Action dated Apr. 7, 2008.
U.S. Appl. No. 10/636,314 Response to Restriction Requirement dated Aug. 5, 2008.
U.S. Appl. No. 10/636,314 Office Action dated Oct. 3, 2008.
U.S. Appl. No. 10/636,314 Response to Office Action dated Apr. 3, 2009.
U.S. Appl. No. 10/636,314 Notice of Non-Compliant Amendment dated May 4, 2009.
U.S. Appl. No. 10/636,314 Amendment to Office Action Response dated Jun. 4, 2009.
U.S. Appl. No. 10/636,314 Notice of Non-Compliant Amendment dated Jun. 12, 2009.
U.S. Appl. No. 10/636,314 Amendment to Office Action dated Jul. 10, 2009.
U.S. Appl. No. 10/636,314 Final Rejection Office Action dated Oct. 21, 2009.
U.S. Appl. No. 10/636,314 Amendment after Final Office Action dated Dec. 14, 2009.
U.S. Appl. No. 10/636,314 Advisory Action dated Jan. 11, 2010.
U.S. Appl. No. 10/636,314 Notice of Non-Compliant Amendment dated Jan. 11, 2010.
U.S. Appl. No. 10/636,314 Supplemental Amendment and Response dated Mar. 13, 2010.
U.S. Appl. No. 10/636,314 Office Action dated May 11, 2010.
U.S. Appl. No. 10/636,314 Amendment and Response dated Sep. 13, 2010.
U.S. Appl. No. 10/636,314 Final Rejection dated Nov. 24, 2010.
U.S. Appl. No. 10/636,314 Notice of Appeal dated May 19, 2011.
U.S. Appl. No. 10/636,314 Amendment and Request for Continued Examination dated Jul. 19, 2011.
U.S. Appl. No. 10/636,314 Notice of Allowance dated Sep. 13, 2011.
U.S. Appl. No. 10/636,314 Notice of Allowance dated Sep. 19, 2011.
U.S. Appl. No. 10/636,314 Issue Notification dated Oct. 19, 2011.

(56) **References Cited**

OTHER PUBLICATIONS

U.S. Appl. No. 09/474,664 Utility Application filed Dec. 29, 1999.
U.S. Appl. No. 09/474,664 Office Action dated Sep. 23, 2002.
U.S. Appl. No. 09/474,664 Amendment and Response dated Feb. 24, 2003.
U.S. Appl. No. 09/474,664 Notice of Allowance dated May 20, 2003.
U.S. Appl. No. 90/010,356 Request for Ex Parte Reexamination dated Dec. 15, 2008.
U.S. Appl. No. 90/010,356 Office Action Granting Reexamination dated Jan. 17, 2009.
U.S. Appl. No. 90/010,356 First Office Action dated Jul. 7, 2009.
U.S. Appl. No. 90/010,356 First Office Action Response dated Sep. 1, 2009.
U.S. Appl. No. 90/010,356 Patent Owner Interview Summary dated Oct. 23, 2009.
U.S. Appl. No. 90/010,356 Office Action Final dated Dec. 4, 2009.
U.S. Appl. No. 90/010,356 Amendment and Response to Office Action dated Dec. 18, 2009.
U.S. Appl. No. 90/010,356 Amendment and Response to Office Action dated Jan. 4, 2010.
U.S. Appl. No. 90/010,356 Advisory Action dated Jan. 21, 2010.
U.S. Appl. No. 90/010,356 Amendment and Response to Advisory Action dated Feb. 8, 2010.
U.S. Appl. No. 90/010,356 Notice of Intent to Issue a Reexam Certificate dated Mar. 2, 2010.
U.S. Appl. No. 90/010,356 Reexamination Certificate Issued dated Jun. 22, 2010.
U.S. Appl. No. 95/000,659 Inter Partes Reexam Request dated Feb. 13, 2012.
U.S. Appl. No. 95/000,659 Order Granting Reexamination dated Apr. 3, 2012.
U.S. Appl. No. 95/000,659 Office Action dated Apr. 3, 2012.
U.S. Appl. No. 95/000,659 Office Action Response dated Jun. 4, 2012 (including Exhibits 1 & 2) (4 documents).
U.S. Appl. No. 95/000,659 Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012.
U.S. Appl. No. 95/000,659 Appendix R-1 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Declaration of Prof. Dr. Bernhard Plattner).
U.S. Appl. No. 95/000,659 Appendix R-2 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Prof. Dr. Bernhard Plattner CV).
U.S. Appl. No. 95/000,659 Appendix R-3 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Listing of Publications to Prof. Dr. Bernhard Plattner updated Feb. 2012).
U.S. Appl. No. 95/000,659 Appendix R-4 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012(Office Action Granting Reexamination in 95/000,660 dated May 10, 2012).
U.S. Appl. No. 95/000,659 Appendix R-5 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Office Action in 95/000,660 dated May 10, 2012).
U.S. Appl. No. 95/000,659 Appendix R-6 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Implicit Networks, Inc. U.S. Pat. No. 6,629,163 Claims Chart).
U.S. Appl. No. 95/000,659 Appendix R-7 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Internet Protocol DARPA Internet Program Protocol Specification dated Sep. 1991).
U.S. Appl. No. 95/000,659 Appendix R-8 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Atkinson, IP Encapsulating Security Payload (ESP) dated Aug. 1995).
U.S. Appl. No. 95/000,659 Appendix R-9 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Claim Construction Order dated Feb. 29, 2012).
U.S. Appl. No. 95/000,659 Appendix R-10-1 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (vol. I of Edward Balassanian Deposition Transcript dated May 30, 2012).
U.S. Appl. No. 95/000,659 Appendix R-10-2 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (vol. II of Edward Balassanian Deposition Transcript dated May 31, 2012).
U.S. Appl. No. 95/000,659 Appendix R-10-3 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (vol. III of Edward Balassanian Deposition Transcript dated Jun. 7, 2012).
U.S. Appl. No. 95/000,659 Appendix R-10-4 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (vol. IV of Edward Balassanian Deposition Transcript dated Jun. 8, 2012).
U.S. Appl. No. 95/000,659 Appendix R-11 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Implicit Networks, Inc.'s Response to Juniper Networks, Inc.'s First Set of Requests for Admission 1-32).
U.S. Appl. No. 95/000,659 Action Closing Prosecution dated Oct. 1, 2012.
U.S. Appl. No. 95/000,659 Petition to Withdraw and Reissue Action Closing Prosecution dated Nov. 20, 2012.
U.S. Appl. No. 95/000,659 Patent Owner Comments to Action Closing Prosecution dated Dec. 3, 2012.
U.S. Appl. No. 95/000,659 Opposition to Petition dated Dec. 17, 2012.
U.S. Appl. No. 95/000,659 Third Party Comments to Action Closing Prosecution dated Jan. 2, 2013.
U.S. Appl. No. 95/000,660 Inter Partes Reexam Request dated Mar. 2, 2012.
U.S. Appl. No. 95/000,660 Order Granting Reexamination dated May 10, 2012.
U.S. Appl. No. 95/000,660 Office Action dated May 10, 2012.
U.S. Appl. No. 95/000,660 Response to Office Action dated Jul. 10, 2012 (including Exhibits 1 and 2).
U.S. Appl. No. 95/000,660 Third Party Comments to Office After Patent Owner's Response dated Aug. 8, 2012 (including Revised Comments).
U.S. Appl. No. 95/000,660 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Declaration of Prof. Dr. Bernhard Plattner).
U.S. Appl. No. 95/000,660 Appendix R-1 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Prof. Dr. Bernhard Plattner CV).
U.S. Appl. No. 95/000,660 Appendix R-3 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Listing of Publications to Prof. Dr. Bernhard Plattner updated Feb. 2012).
U.S. Appl. No. 95/000,660 Appendix R-4 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012(Office Action Granting Reexamination in 95/000,660 dated May 10, 2012).
U.S. Appl. No. 95/000,660 Appendix R-5 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Office Action in 95/000,660 dated May 10, 2012).
U.S. Appl. No. 95/000,660 Appendix R-6 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Implicit Networks, Inc. U.S. Pat. No. 6,629,163 Claims Chart).
U.S. Appl. No. 95/000,660 Appendix R-7 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Internet Protocol DARPA Internet Program Protocol Specification dated Sep. 1991).
U.S. Appl. No. 95/000,660 Appendix R-8 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Atkinson, IP Encapsulating Security Payload (ESP) dated Aug. 1995).
U.S. Appl. No. 95/000,660 Appendix R-9 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Claim Construction Order dated Feb. 29, 2012).

(56) **References Cited**

OTHER PUBLICATIONS

U.S. Appl. No. 95/000,660 Appendix R-10 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (vol. I-IV of Edward Balassanian Deposition Transcript dated May 30, 2012).
U.S. Appl. No. 95/000,660 Appendix R-11 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Shacham, A., et al, "*IP Payload Compression Protocol*", Network Working Group, RFC 3173 Sep. 2001).
U.S. Appl. No. 95/000,660 Appendix R-12 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Shacham, A., et al, "*IP Payload Compression Protocol*", Network Working Group, RFC 2393 Dec. 1998).
U.S. Appl. No. 95/000,660 Appendix R-13 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 ('163 Pfeiffer Claim Chart).
U.S. Appl. No. 95/000,660 Appendix R-14 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Ylonen, T., "*SSH Transport Layer Protocol*", Network Working Group—Draft Feb. 22, 1999).
U.S. Appl. No. 95/000,660 Appendix R-15 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Dommety, G., "*Key and Sequence Number Extensions to GRE*", Network Working Group, RFC 2890 Sep. 2000).
U.S. Appl. No. 95/000,660 Appendix R-16 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Monsour, R., et al, "*Compression in IP Security*" Mar. 1997).
U.S. Appl. No. 95/000,660 Appendix R-17 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Friend, R., Internet Working Group RFC 3943 dated Nov. 2004 *Transport Layer Security Protocol Compression Using Lempel-Ziv-Stac*).
U.S. Appl. No. 95/000,660 Appendix R-18 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012(Implicit Networks, Inc.'s Response to Juniper Networks, Inc.'s First Set of Requests for Admission 1-32).
U.S. Appl. No. 95/000,660 Revised—Third Party Comments to Office After Patent Owner's Response dated Nov. 2, 2012.
U.S. Appl. No. 95/000,660 Action Closing Prosecution dated Dec. 21, 2012.
U.S. Appl. No. 95/000,660 Comments to Action Closing Prosecution dated Feb. 21, 2013 (including Dec of Dr. Ng).
U.S. Appl. No. 95/000,660 Third Party Comments to Action Closing Prosecution dated Mar. 25, 2013.
PCT/US00/33634—PCT application (WO 01/2077 A2—Jul. 12, 2001).
PCT/US00/33634—Written Opinion (WO 01/50277 A3—Feb. 14, 2002).
PCT/US00/33634—International Search Report (Oct. 9, 2001).
PCT/US00/33634—Response to Official Communication dated Dec. 7, 2001 (Mar. 21, 2002).
PCT/US00/33634—International Preliminary Examination Report (Apr. 8, 2002).
PCT/US00/33634—Official Communication (Jan. 24, 2003).
PCT/US00/33634—Response to Official Communication dated Jan. 24, 2003 (Mar. 12, 2003).
PCT/US00/33634—Official Communication (May 13, 2004).
PCT/US00/33634—Response to Summons to Attend Oral Proceeding dated May 13, 2004 (Oct. 9, 2004).
PCT/US00/33634—Decision to Refuse a European Patent application (Nov. 12, 2004).
PCT/US00/33634—Minutes of the oral proceedings before the Examining Division (Oct. 12, 2004).
PCT/US00/33634—Closure of the procedure in respect to Application No. 00984234.5-2212 (Feb. 22, 2005).
May 3, 2013 Expert Report of Dr. Alfonso Cardenas Regarding Validity of U.S. Pat. Nos. 6,877,006; 7,167,864; 7,720,861; and 8,082,268 (6 documents).
Expert Report of Dr. Alfonso Cardenas Regarding Validity of U.S. Pat. No. 7,167,864 (3 documents).
"InfoReports User Guide: Version 3.3.1;" Platinum Technology, Publication No. PRO-X-331-UG00-00, printed Apr. 1998; pp. 1-430.
Non-Final Office Action in Inter Partes Reexamination Control No. 95/000,659 issued Aug. 16, 2013, 107 pages.
Decision on Petition in Reexamination Control No. 95/000,659 issued Aug. 19, 2013, 3 pages.
Response to Non-Final Office Action in Reexamination Control No. 95/000,659 mailed Oct. 2, 2013 including Exhibits A-C, 37 pages.
Decision on Petition in Reexamination Control No. 95/000,660 issued Jul. 30, 2013, 12 pages.
Non-Final Office Action in Inter Partes Reexamination Control No. 95/000,660 issued Aug. 30, 2013, 23 pages.
RFC: 791. Internet Protocol: DARPA Internet Program Protocol Specification, Sep. 1981, prepared for Defense Advanced Research Projects Agency Information Processing Techniques Office by Information Sciences Institute University of Southern California, 52 pages.

* cited by examiner

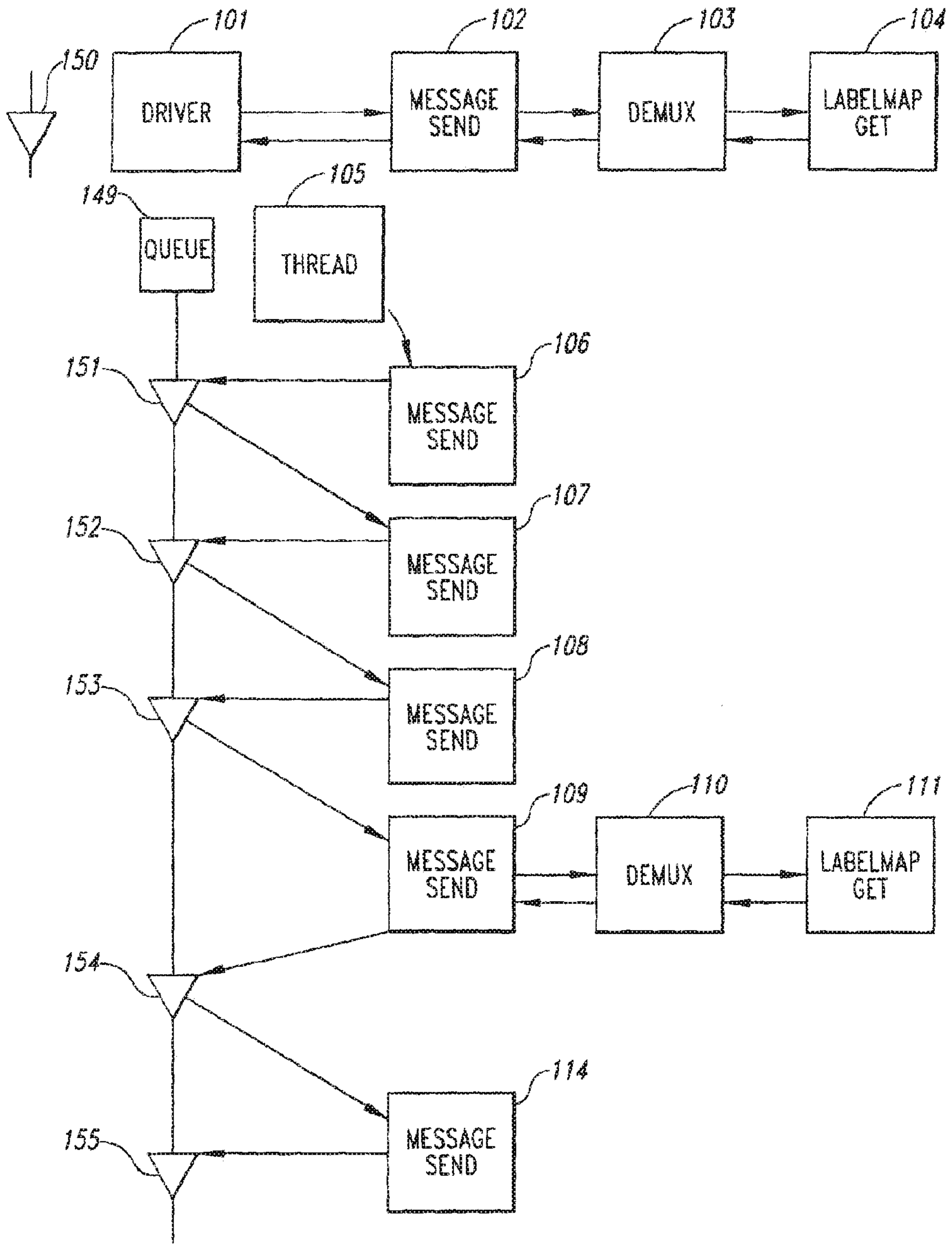

101
DRIVER
102
MESSAGE SEND
103
DEMUX
104
LABELMAP GET
150
149
QUEUE
105
THREAD
106
MESSAGE SEND
151
107
MESSAGE SEND
152
108
MESSAGE SEND
153
109
MESSAGE SEND
110
DEMUX
111
LABELMAP GET
154
114
MESSAGE SEND
155

Fig. 1

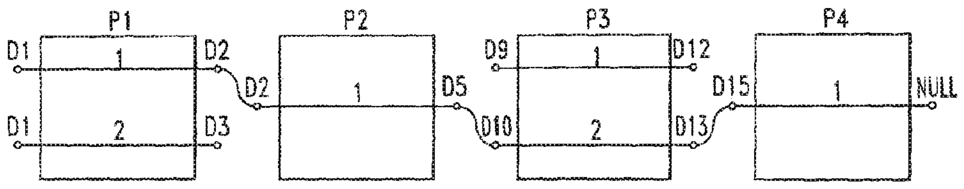

P1
P2
P3
P4
D1
1
D2
D1
2
D3
D2
1
D5
D9
1
D12
D10
2
D13
D15
1
NULL

Fig. 2

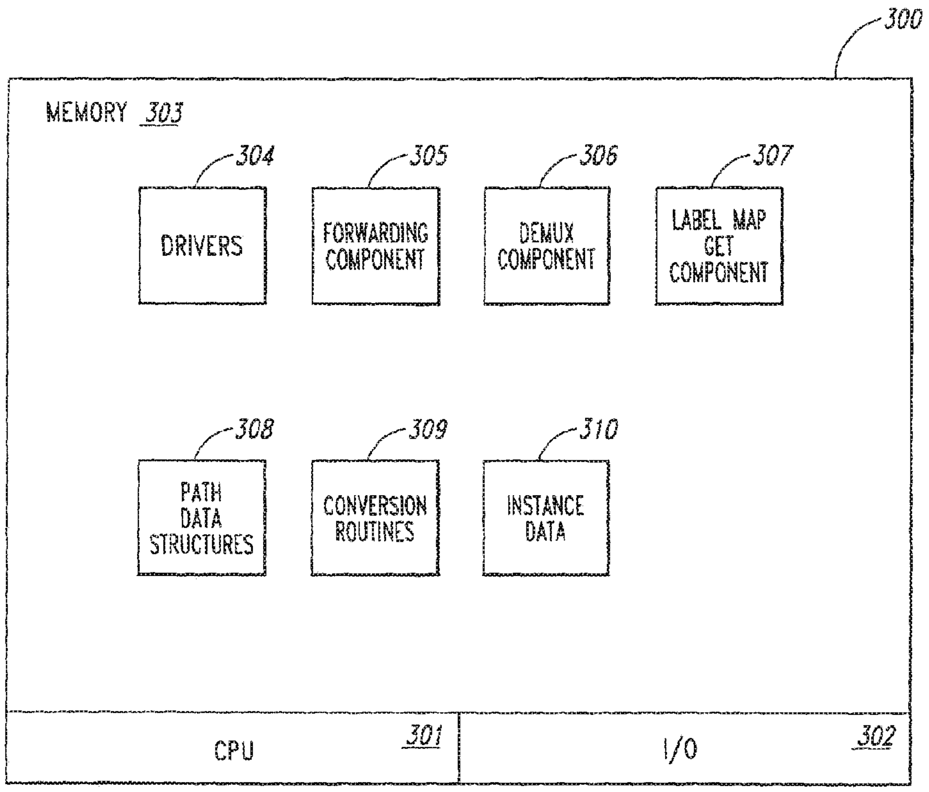

300
MEMORY 303
304
DRIVERS
305
FORWARDING COMPONENT
306
DEMUX COMPONENT
307
LABEL MAP GET COMPONENT
308
PATH DATA STRUCTURES
309
CONVERSION ROUTINES
310
INSTANCE DATA
CPU 301
I/O 302

Fig. 3

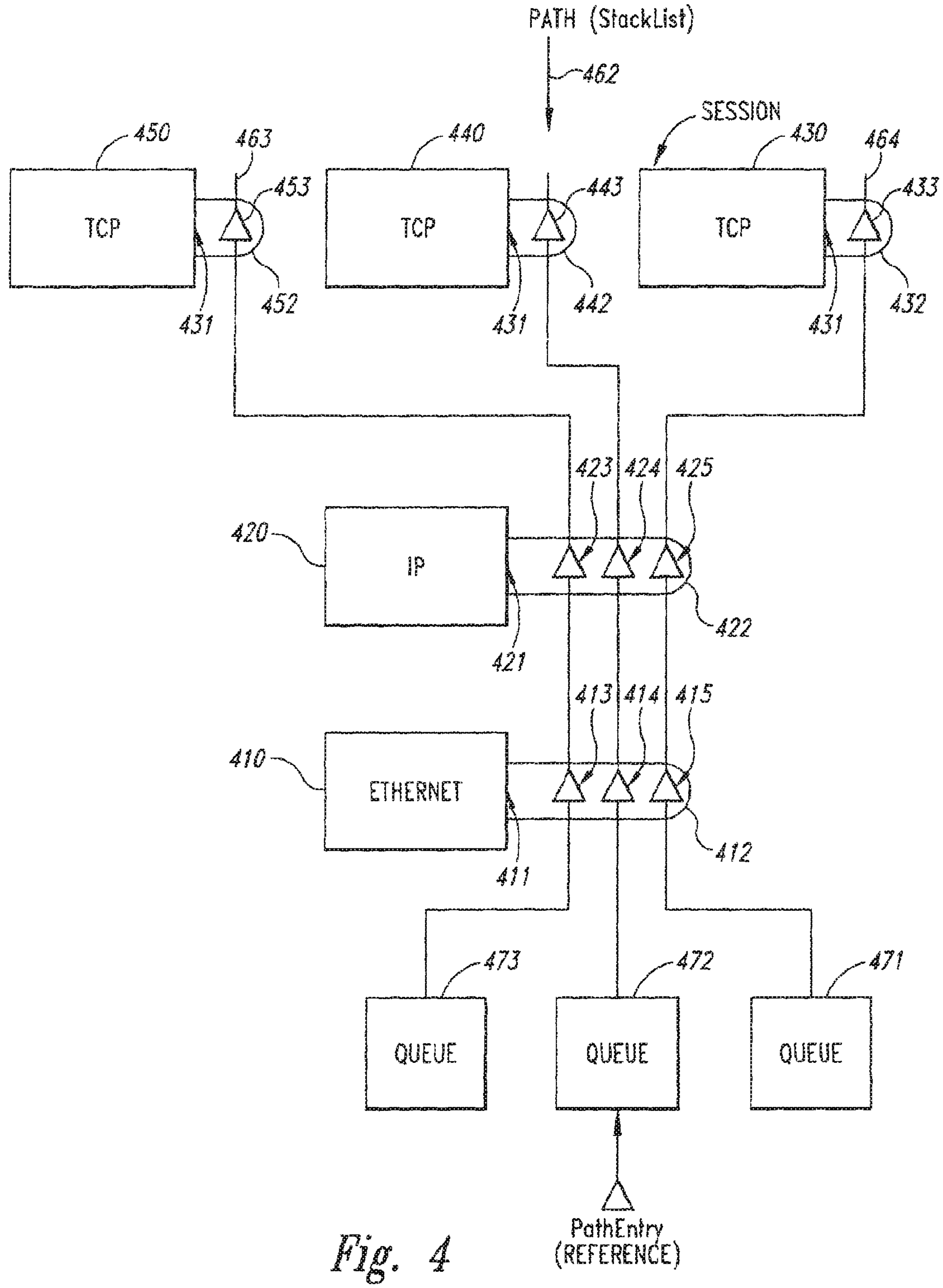

PATH (StackList)
462
SESSION
450
463
440
430
464
TCP
453
TCP
443
TCP
433
431
452
431
442
431
432
423
424
425
420
IP
422
421
413
414
415
410
ETHERNET
412
411
473
472
471
QUEUE
QUEUE
QUEUE
PathEntry (REFERENCE)

Fig. 4

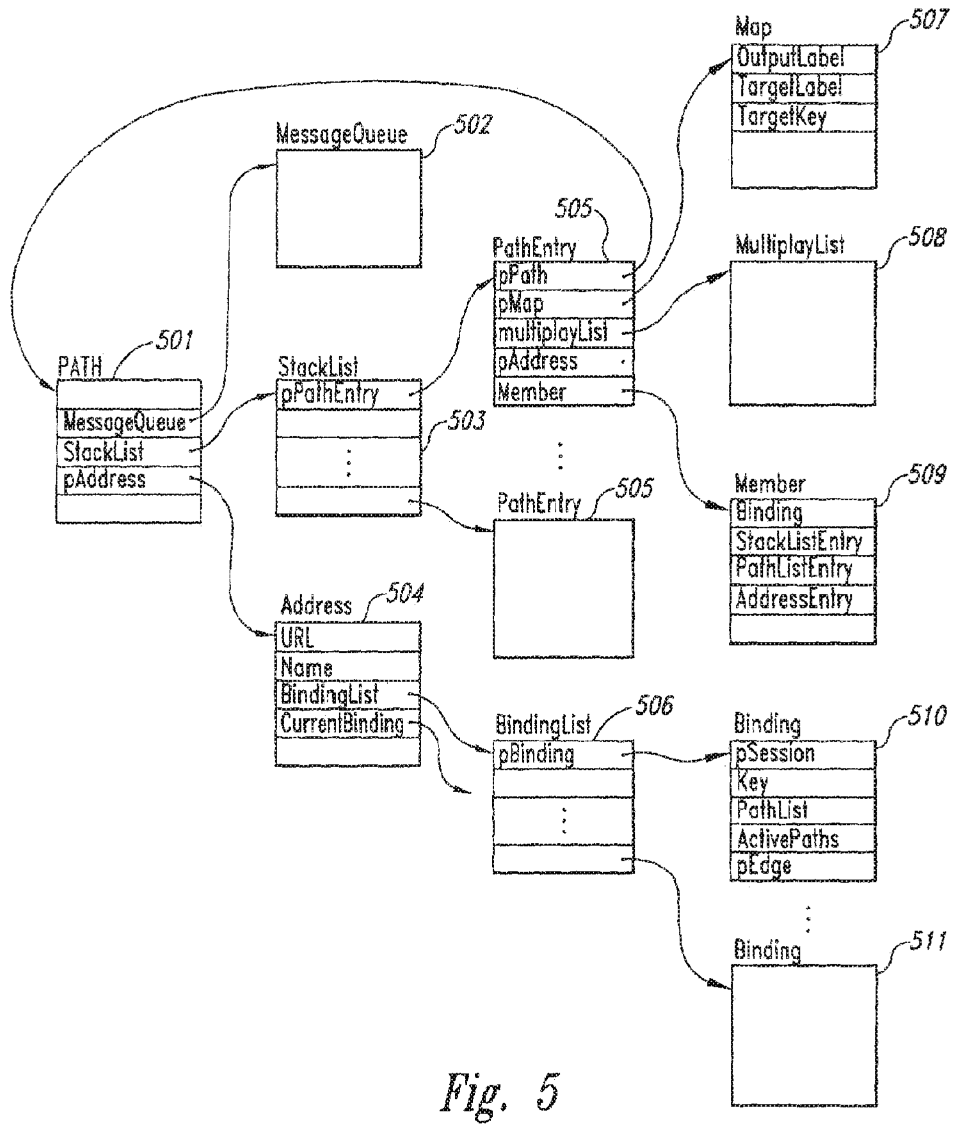

Map
507
OutputLabel
TargetLabel
TargetKey
MessageQueue
502
505
PathEntry
pPath
pMap
multiplayList
pAddress
Member
MultiplayList
508
PATH
501
MessageQueue
StackList
pAddress
StackList
pPathEntry
503
PathEntry
505
Member
509
Binding
StackListEntry
PathListEntry
AddressEntry
Address
504
URL
Name
BindingList
CurrentBinding
BindingList
506
pBinding
Binding
510
pSession
Key
PathList
ActivePaths
pEdge
Binding
511

Fig. 5

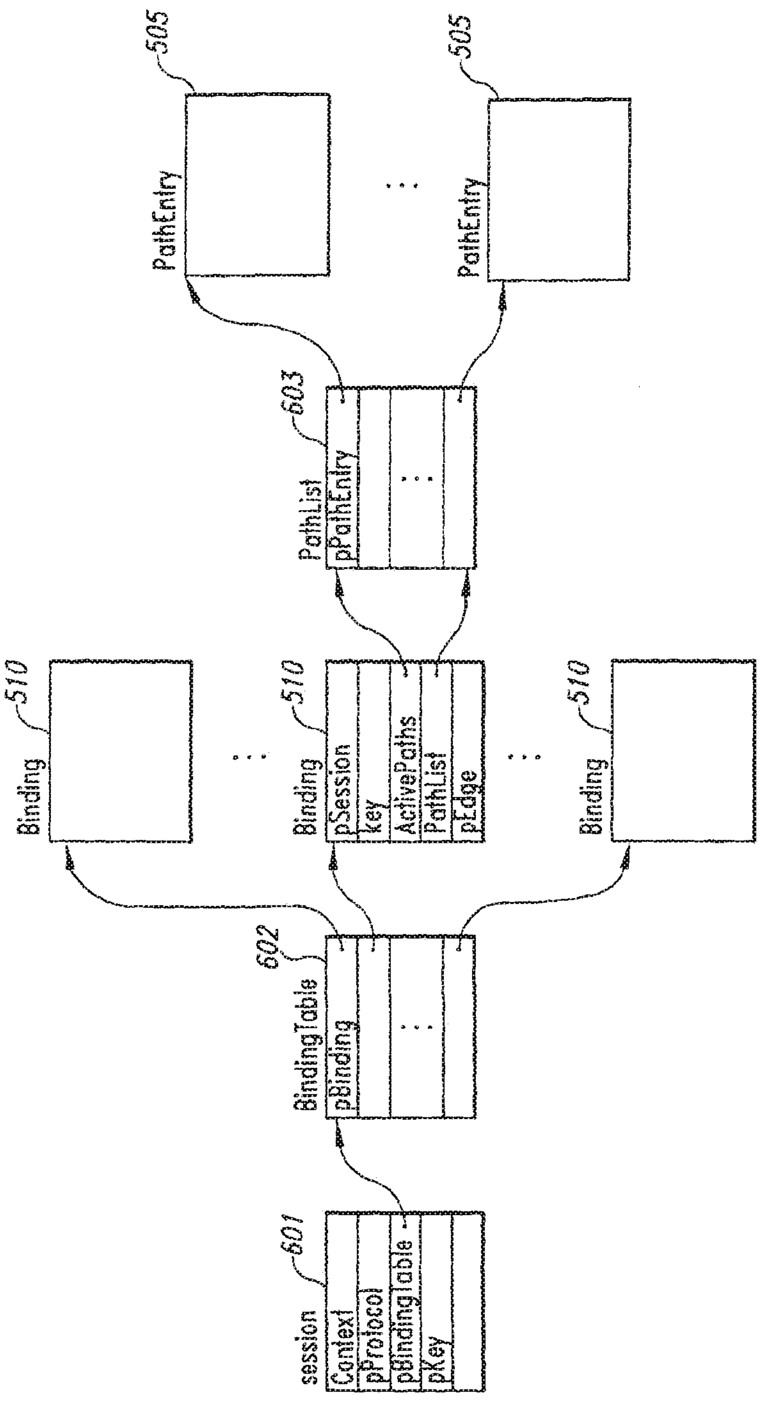

505
PathEntry
505
PathEntry
603
PathList
pPathEntry
510
Binding
510
Binding
pSession
key
ActivePaths
PathList
pEdge
510
Binding
602
BindingTable
pBinding
601
session
Context
pProtocol
pBindingTable
pKey

Fig. 6

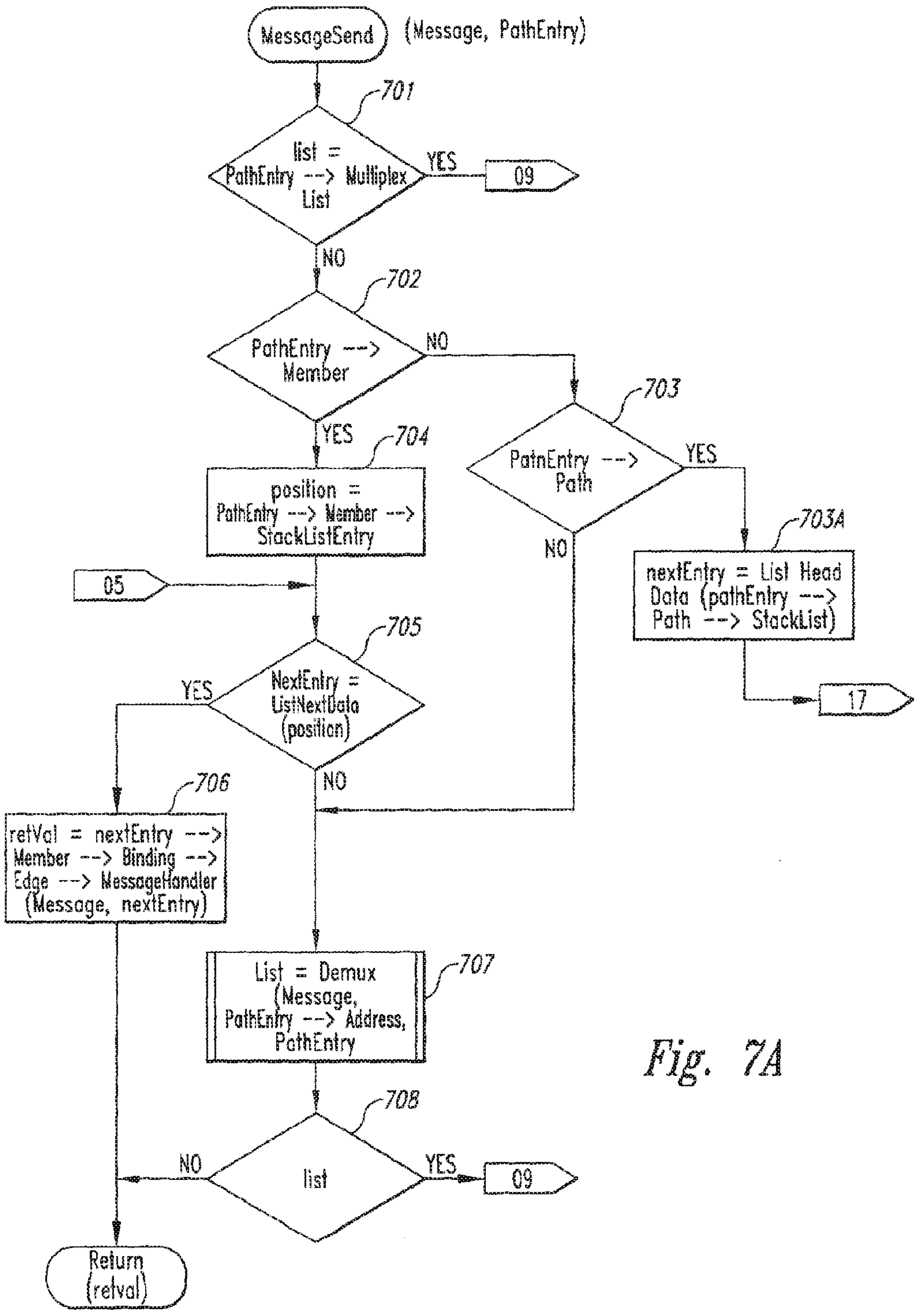

MessageSend
(Message, PathEntry)
701
list = PathEntry --> Multiplex List
YES
09
NO
702
PathEntry --> Member
NO
703
PatnEntry --> Path
YES
703A
nextEntry = List Head Data (pathEntry --> Path --> StackList)
17
NO
YES
704
position = PathEntry --> Member --> StackListEntry
05
705
NextEntry = ListNextData (position)
YES
NO
706
retVal = nextEntry --> Member --> Binding --> Edge --> MessageHandler (Message, nextEntry)
707
List = Demux (Message, PathEntry --> Address, PathEntry
708
list
NO
YES
09
Return (retval)

Fig. 7A

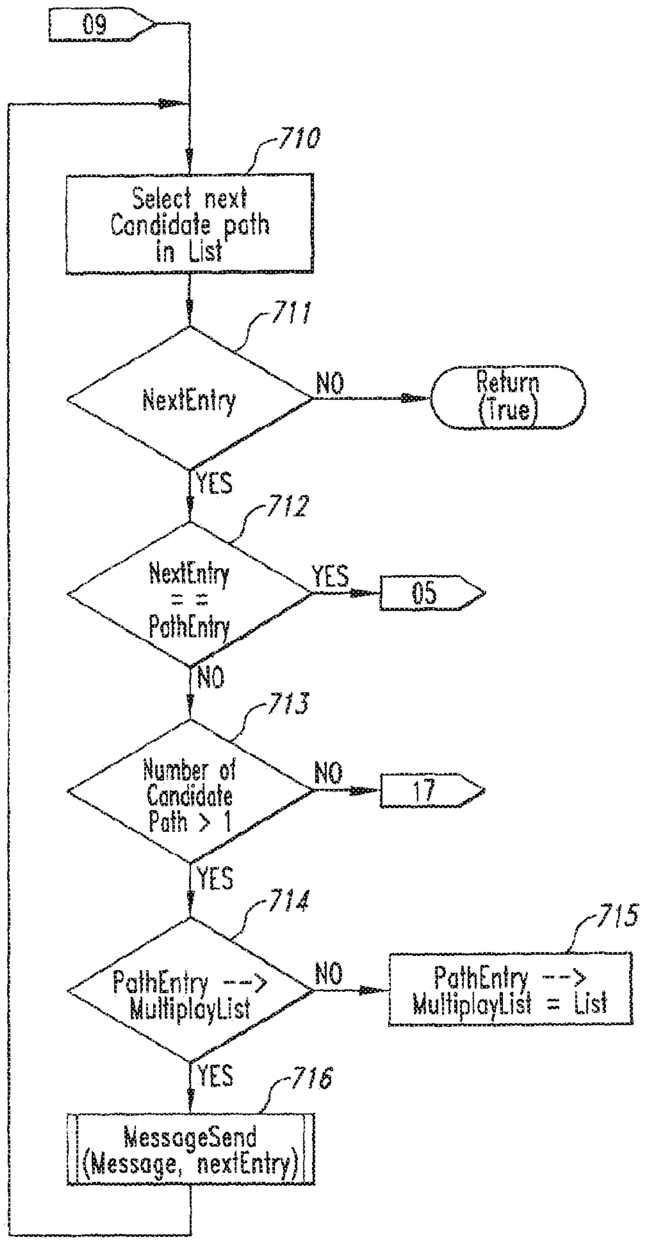

09
710
Select next Candidate path in List
711
NextEntry
NO
Return (True)
YES
712
NextEntry = = PathEntry
YES
05
NO
713
Number of Candidate Path > 1
NO
17
YES
714
PathEntry --> MultiplayList
NO
715
PathEntry --> MultiplayList = List
YES
716
MessageSend (Message, nextEntry)

Fig. 7B

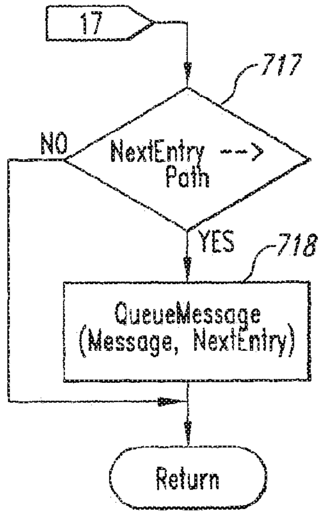

17
717
NO
NextEntry --> Path
YES
718
QueueMessage (Message, NextEntry)
Return

Fig. 7C

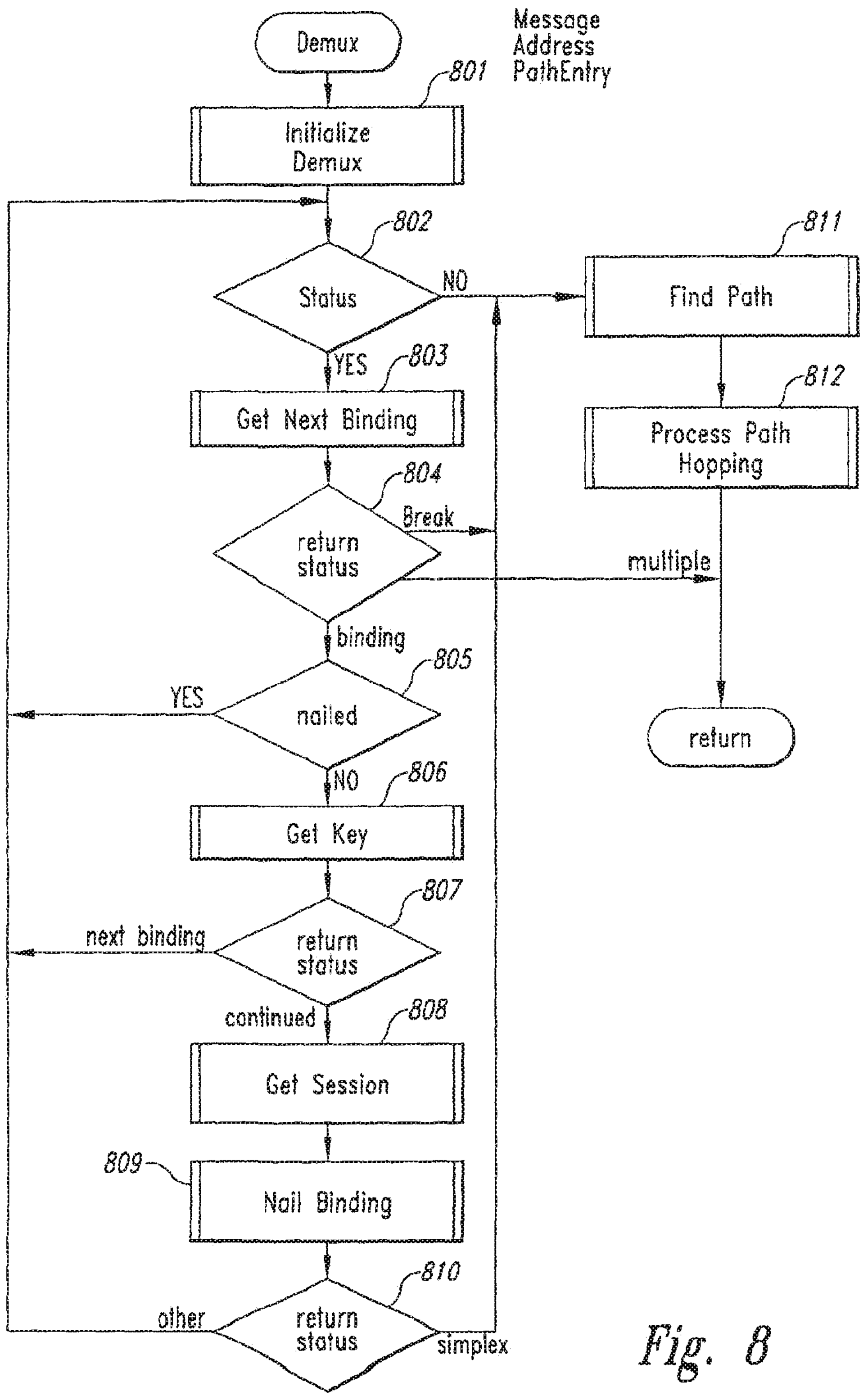

Demux
Message
Address
PathEntry
801
Initialize Demux
802
Status
NO
YES
811
Find Path
812
Process Path Hopping
803
Get Next Binding
804
return status
Break
multiple
binding
805
nailed
YES
NO
return
806
Get Key
807
return status
next binding
continued
808
Get Session
809
Nail Binding
810
return status
other
simplex

Fig. 8

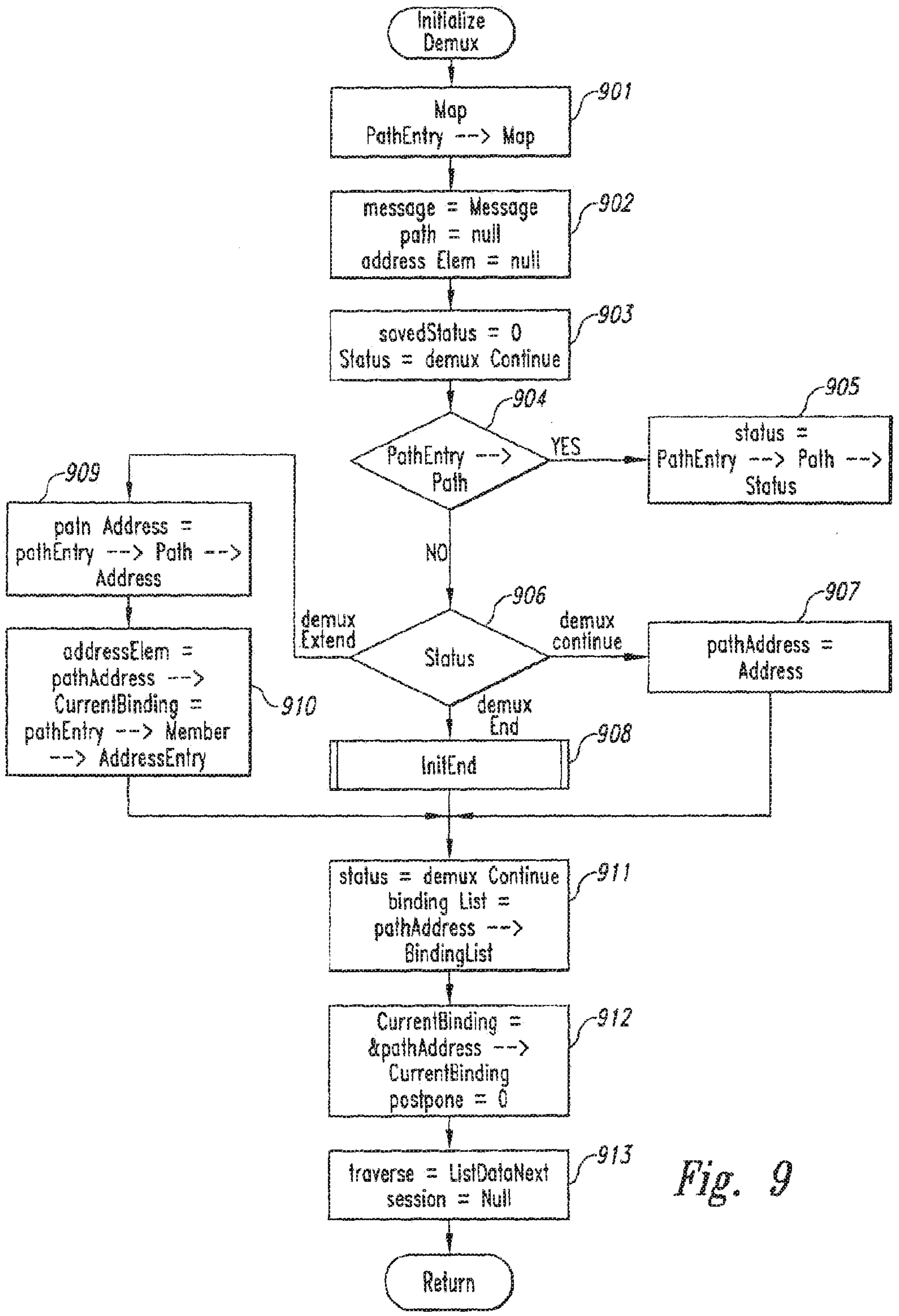

Initialize Demux
901
Map
PathEntry --> Map
902
message = Message
path = null
address Elem = null
903
savedStatus = 0
Status = demux Continue
904
PathEntry --> Path
YES
905
status = PathEntry --> Path --> Status
NO
906
Status
demux Extend
demux continue
demux End
907
pathAddress = Address
908
InitEnd
909
path Address = pathEntry --> Path --> Address
910
addressElem = pathAddress --> CurrentBinding = pathEntry --> Member --> AddressEntry
911
status = demux Continue
binding List = pathAddress --> BindingList
912
CurrentBinding = &pathAddress --> CurrentBinding
postpone = 0
913
traverse = ListDataNext
session = Null
Return

Fig. 9

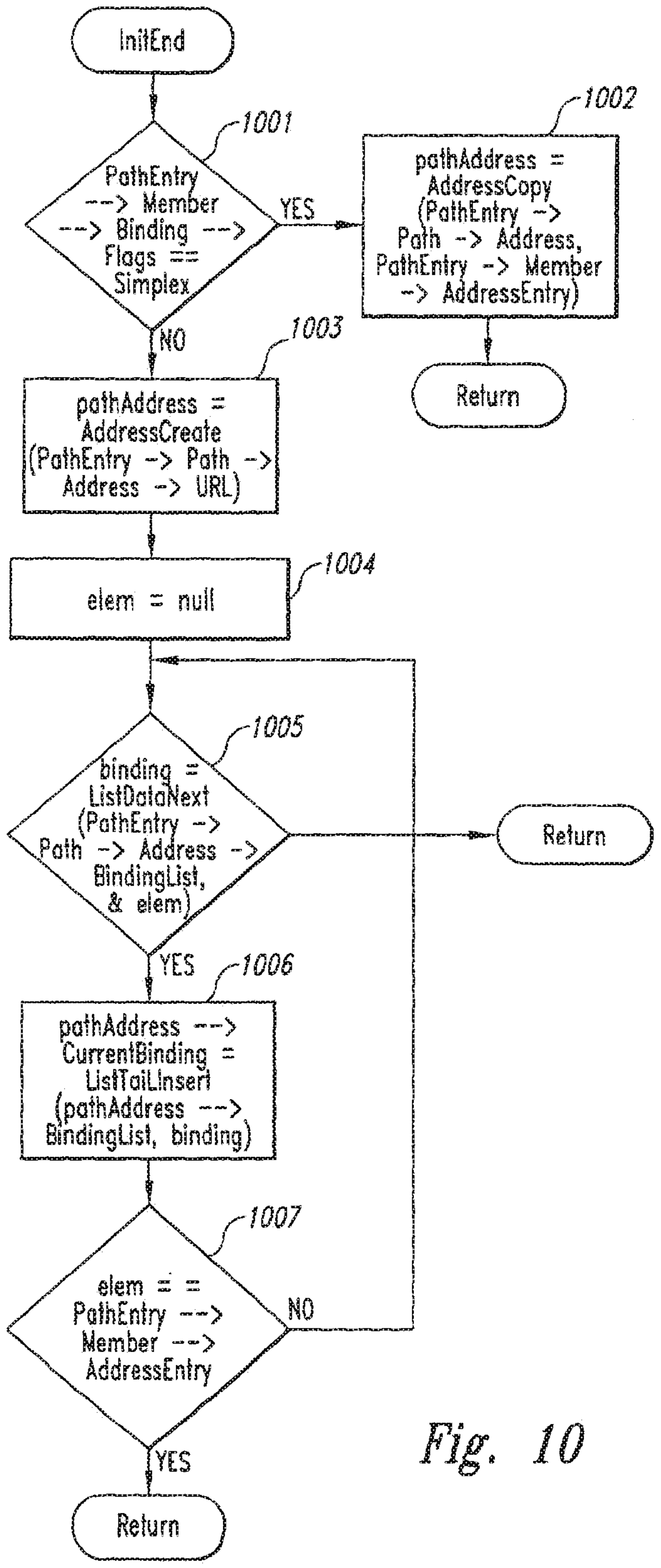

InitEnd
1001
PathEntry --> Member --> Binding --> Flags == Simplex
YES
NO
1002
pathAddress = AddressCopy (PathEntry -> Path -> Address, PathEntry -> Member -> AddressEntry)
Return
1003
pathAddress = AddressCreate (PathEntry -> Path -> Address -> URL)
1004
elem = null
1005
binding = ListDataNext (PathEntry -> Path -> Address -> BindingList, & elem)
Return
YES
1006
pathAddress --> CurrentBinding = ListTailInsert (pathAddress --> BindingList, binding)
1007
elem = = PathEntry --> Member --> AddressEntry
NO
YES
Return

Fig. 10

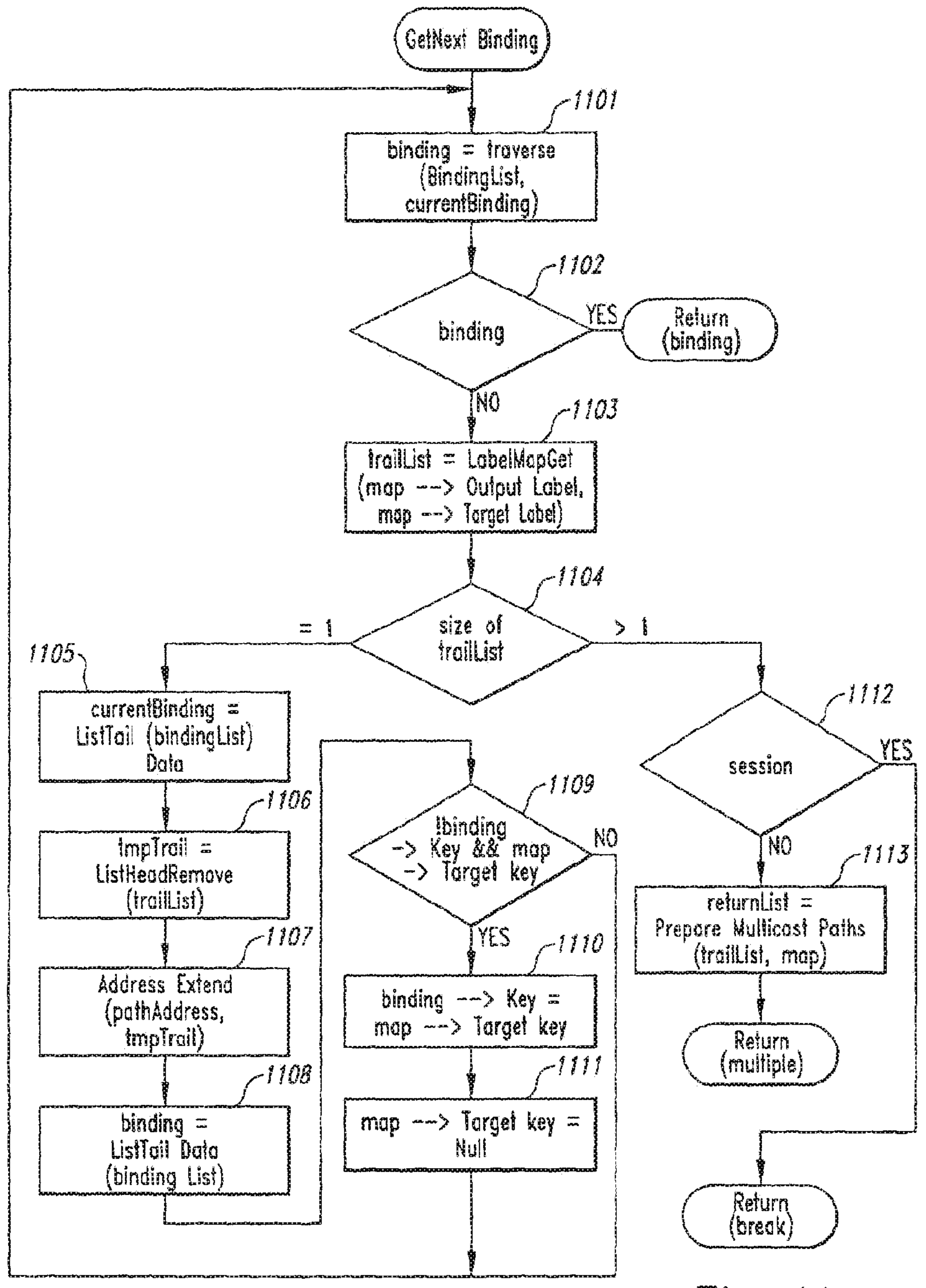

GetNext Binding
1101
binding = traverse (BindingList, currentBinding)
1102
binding
YES
Return (binding)
NO
1103
trailList = LabelMapGet (map --> Output Label, map --> Target Label)
1104
size of trailList
= 1
> 1
1105
currentBinding = ListTail (bindingList) Data
1106
tmpTrail = ListHeadRemove (trailList)
1107
Address Extend (pathAddress, tmpTrail)
1108
binding = ListTail Data (binding List)
1109
!binding -> Key && map -> Target key
NO
YES
1110
binding --> Key = map --> Target key
1111
map --> Target key = Null
1112
session
YES
NO
1113
returnList = Prepare Multicast Paths (trailList, map)
Return (multiple)
Return (break)

Fig. 11

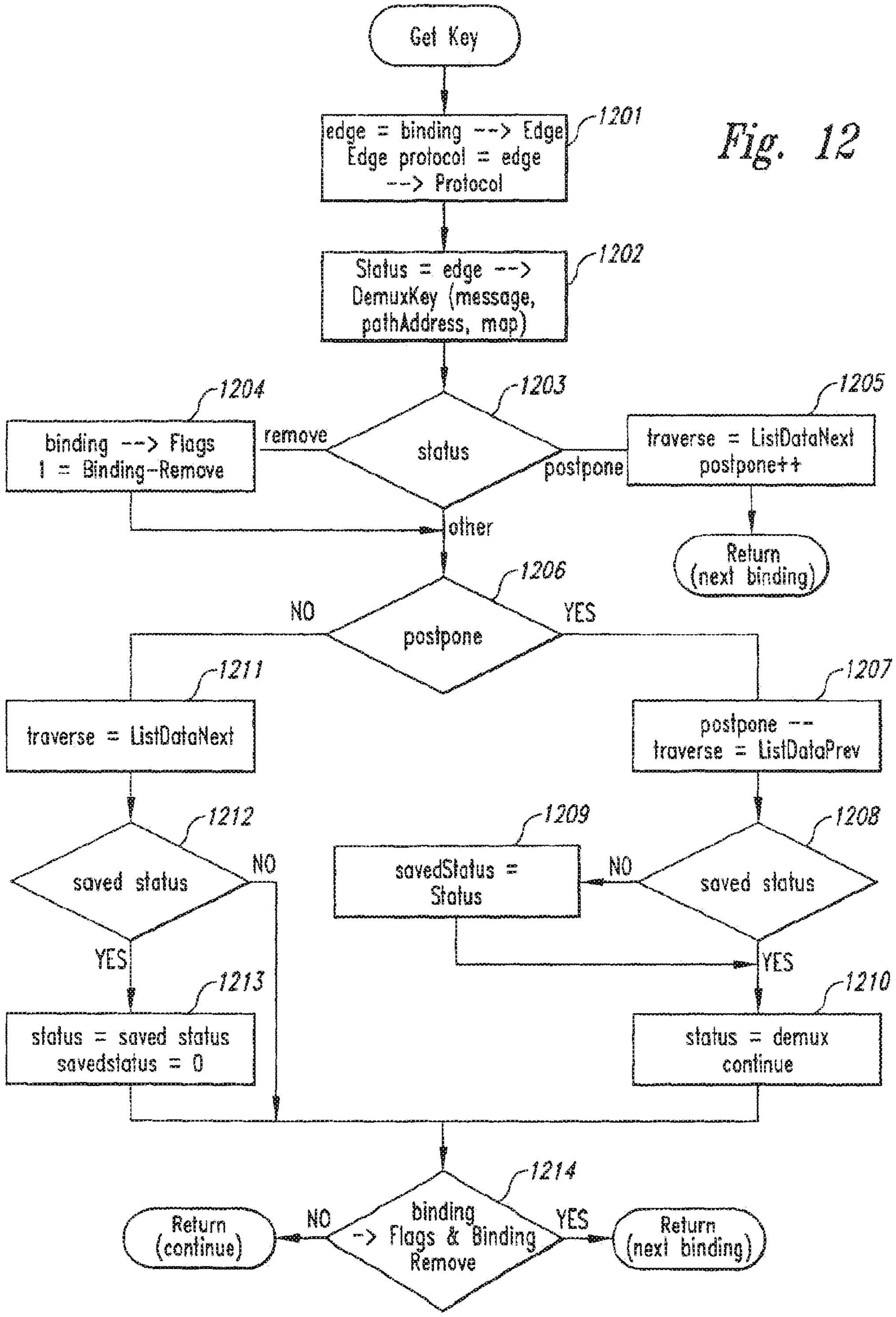

Get Key
1201
edge = binding --> Edge
Edge protocol = edge --> Protocol
1202
Status = edge --> DemuxKey (message, pathAddress, map)
1203
status
remove
postpone
other
1204
binding --> Flags 1 = Binding-Remove
1205
traverse = ListDataNext postpone++
Return (next binding)
1206
postpone
NO
YES
1211
traverse = ListDataNext
1207
postpone --
traverse = ListDataPrev
1212
saved status
NO
YES
1213
status = saved status
savedstatus = 0
1209
savedStatus = Status
1208
saved status
NO
YES
1210
status = demux
continue
1214
binding -> Flags & Binding Remove
NO
YES
Return (continue)
Return (next binding)

Fig. 12

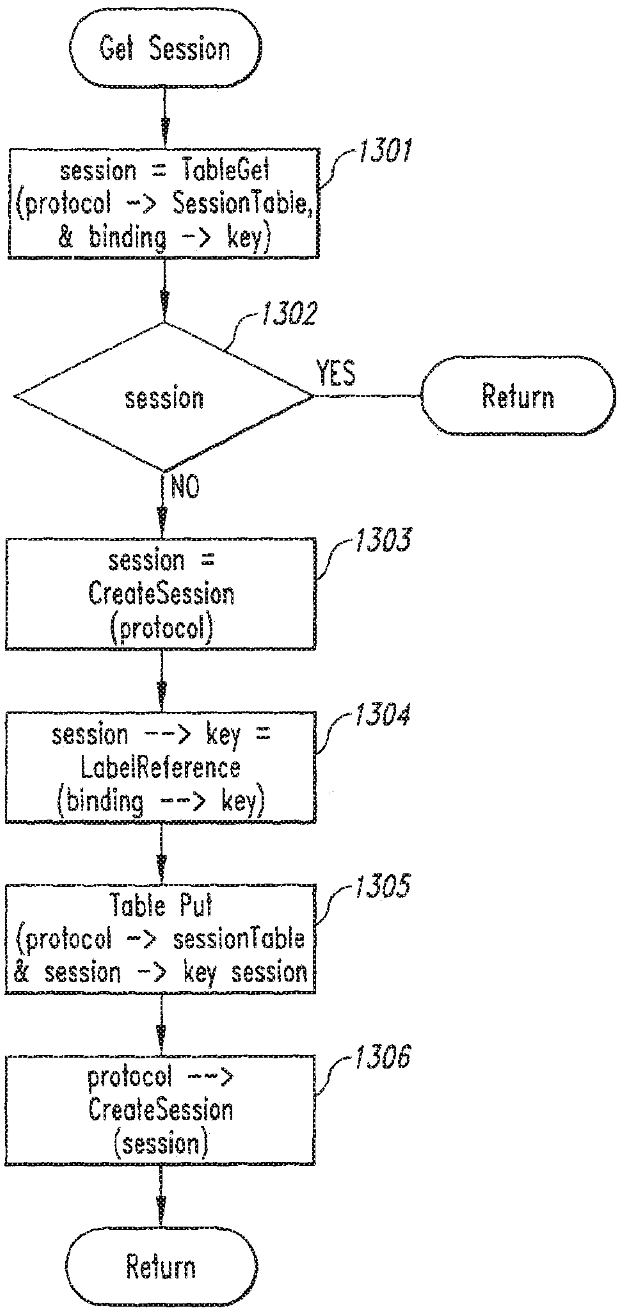

Get Session
1301
session = TableGet (protocol -> SessionTable, & binding -> key)
1302
session
YES
Return
NO
1303
session = CreateSession (protocol)
1304
session --> key = LabelReference (binding --> key)
1305
Table Put (protocol -> sessionTable & session -> key session
1306
protocol --> CreateSession (session)
Return

Fig. 13

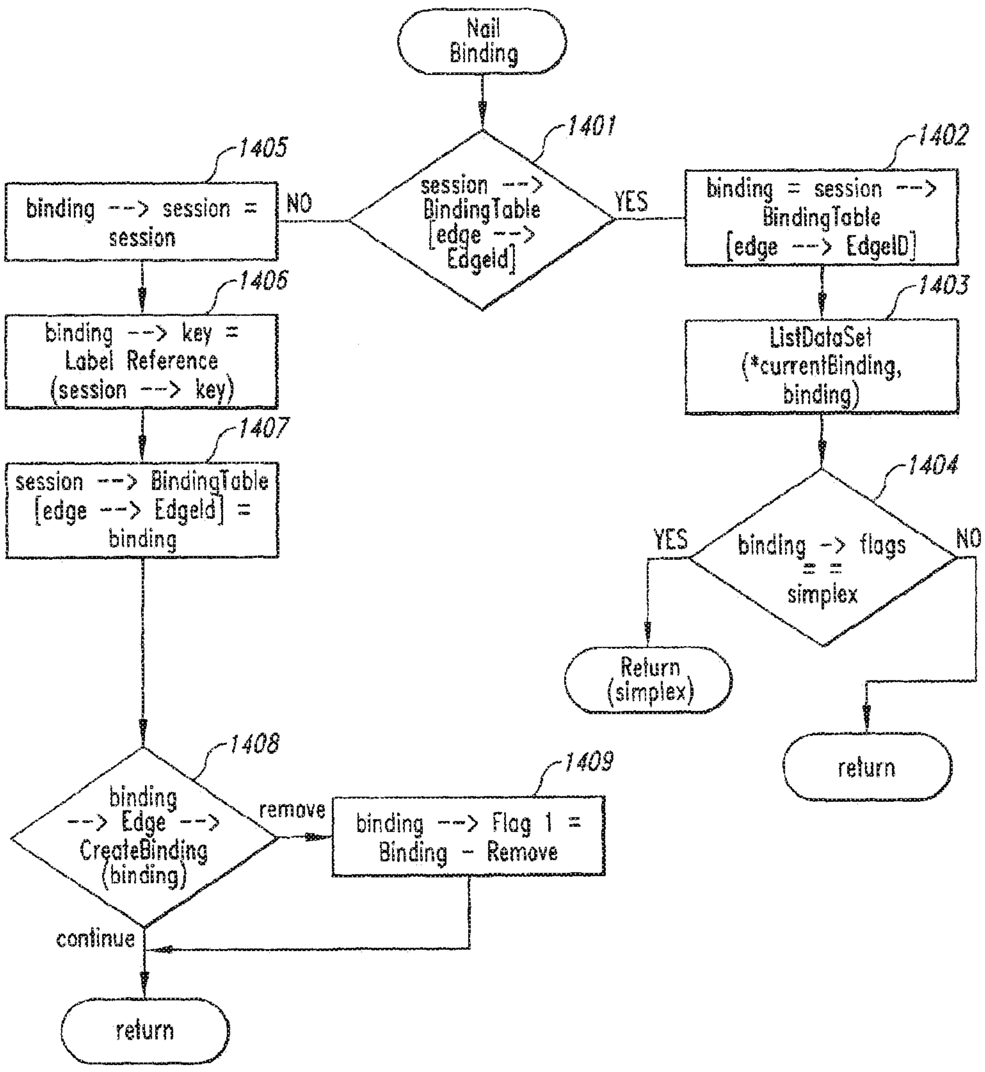

Nail Binding
1401
session --> BindingTable [edge --> EdgeId]
YES
NO
1402
binding = session --> BindingTable [edge --> EdgeID]
1403
ListDataSet (*currentBinding, binding)
1404
binding -> flags = = simplex
YES
NO
Return (simplex)
return
1405
binding --> session = session
1406
binding --> key = Label Reference (session --> key)
1407
session --> BindingTable [edge --> EdgeId] = binding
1408
binding --> Edge --> CreateBinding (binding)
remove
1409
binding --> Flag 1 = Binding - Remove
continue
return

Fig. 14

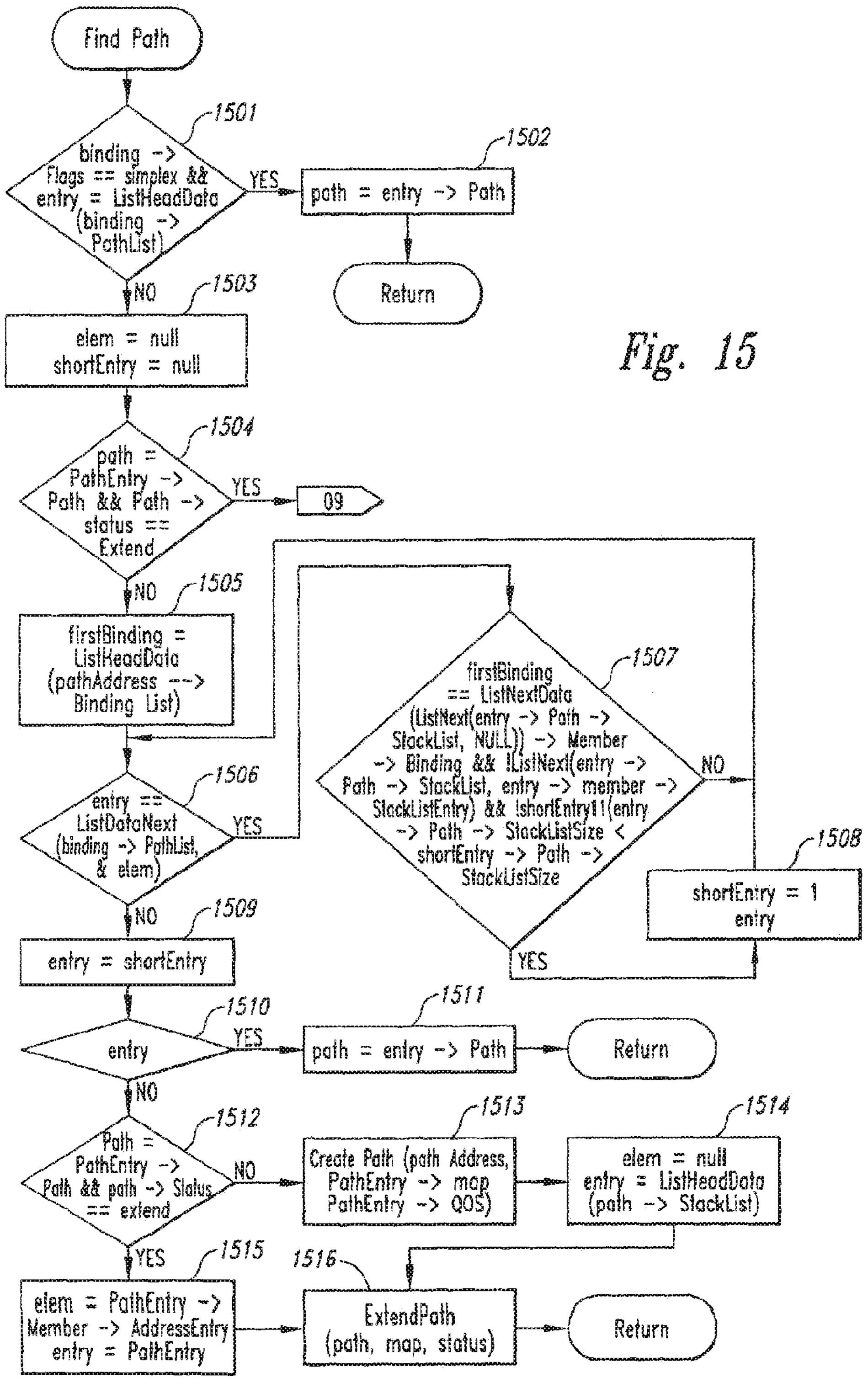
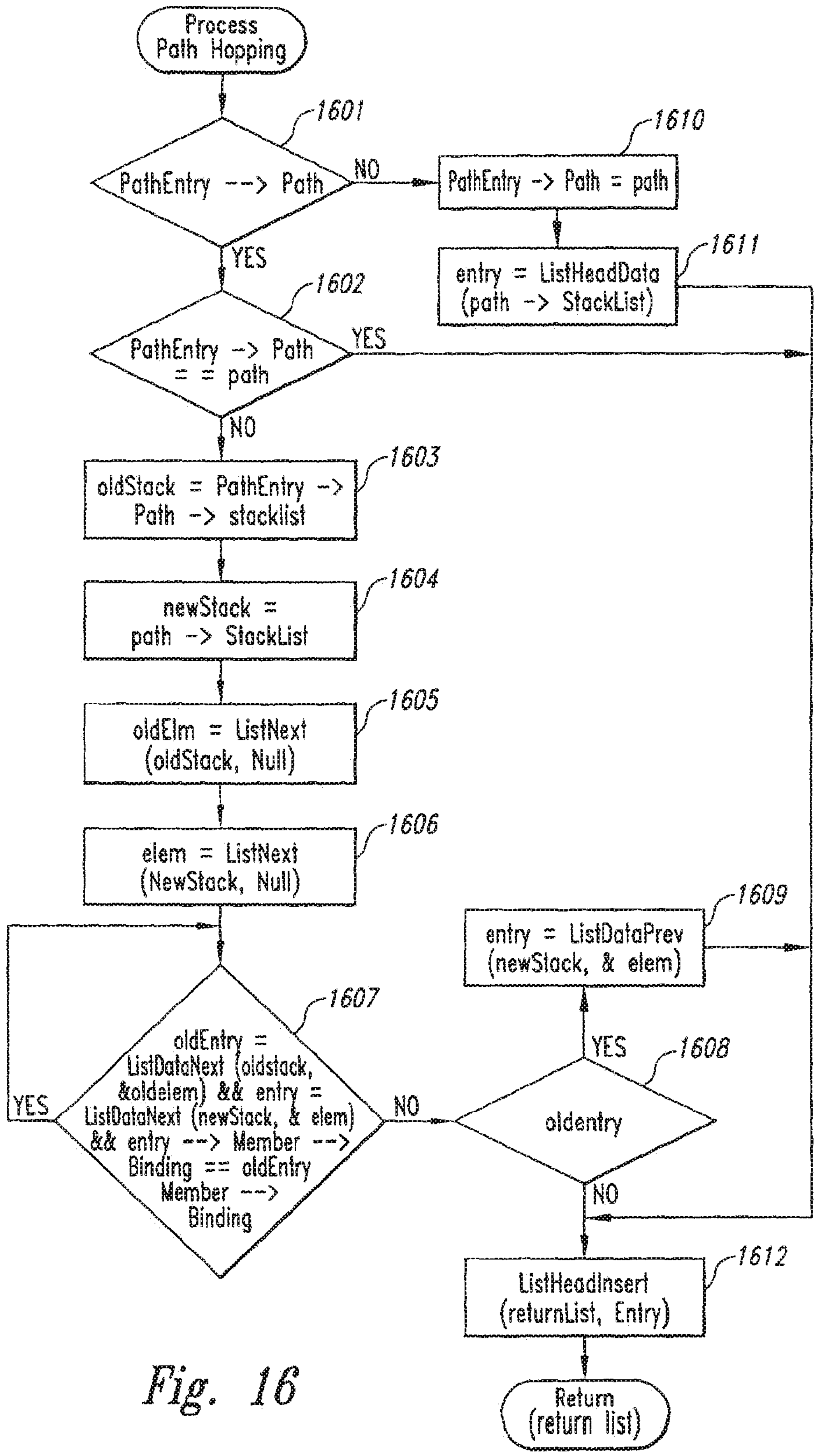

Find Path
1501
binding -> Flags == simplex && entry = ListHeadData (binding -> PathList)
YES
1502
path = entry -> Path
Return
NO
1503
elem = null shortEntry = null
1504
path = PathEntry -> Path && Path -> status == Extend
YES
09
NO
1505
firstBinding = ListHeadData (pathAddress --> Binding List)
1506
entry == ListDataNext (binding -> PathList, & elem)
YES
1507
firstBinding == ListNextData (ListNext(entry -> Path -> StackList, NULL)) -> Member -> Binding && !ListNext(entry -> Path -> StackList, entry -> member -> StackListEntry) && !shortEntry11(entry -> Path -> StackListSize < shortEntry -> Path -> StackListSize
NO
1508
shortEntry = 1 entry
YES
NO
1509
entry = shortEntry
1510
entry
YES
1511
path = entry -> Path
Return
NO
1512
Path = PathEntry -> Path && path -> Status == extend
NO
1513
Create Path (path Address, PathEntry -> map PathEntry -> QOS)
1514
elem = null entry = ListHeadData (path -> StackList)
YES
1515
elem = PathEntry -> Member -> AddressEntry entry = PathEntry
1516
ExtendPath (path, map, status)
Return

Fig. 15



Process Path Hopping
1601
PathEntry --> Path
NO
YES
1610
PathEntry -> Path = path
1611
entry = ListHeadData (path -> StackList)
1602
PathEntry -> Path = = path
YES
NO
1603
oldStack = PathEntry -> Path -> stacklist
1604
newStack = path -> StackList
1605
oldElm = ListNext (oldStack, Null)
1606
elem = ListNext (NewStack, Null)
1609
entry = ListDataPrev (newStack, & elem)
1607
oldEntry = ListDataNext (oldstack, &oldelem) && entry = ListDataNext (newStack, & elem) && entry --> Member --> Binding == oldEntry Member --> Binding
YES
NO
YES
1608
oldentry
NO
1612
ListHeadInsert (returnList, Entry)
Return (return list)

Fig. 16

METHOD AND SYSTEM FOR DATA DEMULTIPLEXING

CROSS REFERENCES TO RELATED APPLICATIONS

The present application is a continuation of U.S. application Ser. No. 14/230,952, filed Mar. 31, 2014 (now U.S. Pat. No. 9,270,790), which is a continuation of U.S. application Ser. No. 13/911,324, filed Jun. 6, 2013 (now U.S. Pat. No. 8,694,683), which is a continuation of U.S. application Ser. No. 13/236,090, filed Sep. 19, 2011 (now abandoned), which is a continuation of US. application Ser. No. 10/636,314, filed Aug. 6, 2003 (now U.S. Pat. No. 8,055,786), which is a continuation of U.S application Ser. No. 09/474,664, filed Dec. 29, 1999 (now U.S. Pat. No. 6,629,163); the disclosures of each of the above-referenced applications are incorporated by reference herein in their entireties.

TECHNICAL FIELD

The present invention relates generally to a computer system for data demultiplexing.

BACKGROUND

Computer systems, which are becoming increasingly pervasive, generate data in a wide variety of formats. The Internet is an example of interconnected computer systems that generate data in many different formats. Indeed, when data is generated on one computer system and is transmitted to another computer system to be displayed, the data may be converted in many different intermediate formats before it is eventually displayed. For example, the generating computer system may initially store the data in a bitmap format. To send the data to another computer system, the computer system may first compress the bitmap data and then encrypt the compressed data. The computer system may then convert that compressed data into a TCP format and then into an IP format. The IP formatted data may be converted into a transmission format, such as an ethernet format. The data in the transmission format is then sent to a receiving computer system. The receiving computer system would need to perform each of these conversions in reverse order to convert the data in the bitmap format. In addition, the receiving computer system may need to convert the bitmap data into a format that is appropriate for rendering on output device.

In order to process data in such a wide variety of formats, both sending and receiving computer systems need to have many conversion routines available to support the various formats. These computer systems typically use predefined configuration information to load the correct combination of conversion routines for processing data. These computer systems also use a process-oriented approach when processing data with these conversion routines. When using a process-oriented approach, a computer system may create a separate process for each conversion that needs to take place. A computer system in certain situations, however, can be expected to receive data and to provide data in many different formats that may not be known until the data is received. The overhead of statically providing each possible series of conversion routines is very high. For example, a computer system that serves as a central controller for data received within a home would be expected to process data received via telephone lines, cable TV lines, and satellite connections in many different formats. The central controller

would be expected to output the data to computer displays, television displays, entertainment centers, speakers, recording devices, and so on in many different formats. Moreover, since the various conversion routines may be developed by different organizations, it may not be easy to identify that the output format of one conversion routine is compatible with the input format of another conversion routine.

It would be desirable to have a technique for dynamically identifying a series of conversion routines for processing data. In addition, it would be desirable to have a technique in which the output format of one conversion routine can be identified as being compatible with the input format of another conversion routine. It would also be desirable to store the identification of a series of conversion routines so that the series can be quickly identified when data is received.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a block diagram illustrating example processing of a message by the conversion system.

FIG. **2** is a block diagram illustrating a sequence of edges.

FIG. **3** is a block diagram illustrating components of the conversion system in one embodiment.

FIG. **4** is a block diagram illustrating example path data structures in one embodiment.

FIG. **5** is a block diagram that illustrates the interrelationship of the data structures of a path.

FIG. **6** is a block diagram that illustrates the interrelationship of the data structures associated with a session.

FIGS. **7** A, **7**B, and **7**C comprise a flow diagram illustrating the processing of the message send routine.

FIG. **8** is a flow diagram of the demux routine.

FIG. **9** is a flow diagram of the initialize demux routine.

FIG. **10** is a flow diagram of the init end routine.

FIG. **11** is a flow diagram of a routine to get the next binding.

FIG. **12** is a flow diagram of the get key routine.

FIG. **13** is a flow diagram of the get session routine.

FIG. **14** is a flow diagram of the nail binding routine.

FIG. **15** is a flow diagram of the find path routine.

FIG. **16** is a flow diagram of the process of path hopping routine.

DETAILED DESCRIPTION

A method and system for converting a message that may contain multiple packets from an source format into a target format. When a packet of a message is received, the conversion system in one embodiment searches for and identifies a sequence of conversion routines (or more generally message handlers) for processing the packets of the message by comparing the input and output formats of the conversion routines. (A message is a collection of data that is related in some way, such as stream of video or audio data or an email message.) The identified sequence of conversion routines is used to convert the message from the source format to the target format using various intermediate formats. The conversion system then queues the packet for processing by the identified sequence of conversion routines. The conversion system stores the identified sequence so that the sequence can be quickly found (without searching) when the next packet in the message is received. When subsequent packets of the message are received, the conversion system identifies the sequence and queues the packets for pressing by the sequence. Because the conversion system receives multiple messages with different source and target formats and identifies a sequence of conversion routines for each message, the conversion systems effectively "demultiplexes" the messages. That is, the conversion system demultiplexes the messages by receiving the message, identifying the sequence of conversion routines, and controlling the processing of each message by the identified sequence. Moreover, since the conversion routines may need to retain state information between the receipt of one packet of a message and the next packet of that message, the conversion system maintains state information as an instance or session of the conversion routine. The conversion system routes all packets for a message through the same session of each conversion routine so that the same state or instance information can be used by all packets of the message. A sequence of sessions of conversion routines is referred to as a "path." In one embodiment, each path has a path thread associated with it for processing of each packet destined for that path.

In one embodiment, the packets of the messages are initially received by "drivers," such as an Ethernet driver. When a driver receives a packet, it forwards the packet to a forwarding component of the conversion system. The forwarding component is responsible for identifying the session of the conversion routine that should next process the packet and invoking that conversion routine. When invoked by a driver, the forwarding component may use a demultiplexing ("demux") component to identify the session of the first conversion routine of the path that is to process the packet and then queues the packet for processing by the path. A path thread is associated with each path. Each path thread is responsible for retrieving packets from the queue of its path and forwarding the packets to the forwarding component. When the forwarding component is invoked by a path thread, it initially invokes the first conversion routine in the path. That conversion routine processes the packet and forwards the processed packet to the forwarding component, which then invokes the second conversion routine in the path. The process of invoking the conversion routines and forwarding the processed packet to the next conversion routine continues until the last conversion routine in the path is invoked. A conversion routine may defer invocation of the forwarding component until it aggregates multiple packets or may invoke the forwarding component multiple times for a packet once for each sub-packet.

The forwarding component identifies the next conversion routine in the path using the demux component and stores that identification so that the forwarding component can quickly identify the conversion routine when subsequent packets of the same message are received. The demux component, searches for the conversion routine and session that is to next process a packet. The demux component then stores the identification of the session and conversion routine as part of a path data structure so that the conversion system does not need to search for the session and conversion routine when requested to demultiplex subsequent packets of the same message. When searching for the next conversion routine, the demux component invokes a label map get component that identifies the next conversion routine. Once the conversion routine is found, the demux component identifies the session associated with that message by, in one embodiment, invoking code associated with the conversion routine. In general, the code of the conversion routine determines what session should be associated with a message. In certain situations, multiple messages may share the same session. The demux component then extends the path for processing that packet to include that session and conversion routine. The sessions are identified so that each packet is associated with the appropriate state information. The dynamic identification of conversion routines is described in U.S. patent application Ser. No. 11/933,093, filed on Oct. 31, 2007 (now U.S. Pat. No. 7,730,211), entitled "Method and System for Generating a Mapping Between Types of Data," which is hereby incorporated by reference.

FIG. **1** is a block diagram illustrating example processing of a message by the conversion system. The driver **101** receives the packets of the message from a network. The driver performs any appropriate processing of the packet and invokes a message send routine passing the processed packet along with a reference path entry **150**. The message send routine is an embodiment of the forwarding component. A path is represented by a series of path entries, which are represented by triangles. Each member path entry represents a session and conversion routine of the path, and a reference path entry represents the overall path. The passed reference path entry **150** indicates to the message send routine that it is being invoked by a driver. The message send routine invokes the demux routine **102** to search for and identify the path of sessions that is to process the packet. The demux routine may in turn invoke the label map get routine **104** to identify a sequence of conversion routines for processing the packet. In this example, the label map get routine identifies the first three conversion routines, and the demux routine creates the member path entries **151**, **152**, **153** of the path for these conversion routines. Each path entry identifies a session for a conversion routine, and the sequence of path entries **151-155** identifies a path. The message send routine then queues the packet on the queue **149** for the path that is to process the packets of the message. The path thread **105** for the path retrieves the packet from the queue and invokes the message send routine **106** passing the packet and an indication of the path. The message send routine determines that the next session and conversion routine as indicated by path entry **151** has already been found. The message send routine then invokes the instance of the conversion routine for the session. The conversion routine processes the packet and then invokes the message send routine **107**. This processing continues until the message send routine invokes the demux routine **110** after the packet is processed by the conversion routine represented by path entry **153**. The demux routine examines the path and determines that it has no more path entries. The demux routine then invokes the label map get routine **111** to identify the conversion routines for further processing of the packet. When the conversion routines are identified, the demux routine adds path entries **154**, **155** to the path. The messages send routine invokes the conversion routine associated with path entry **154**. Eventually, the conversion routine associated with path entry **155** performs the final processing for the path.

The label map get routine identifies a sequence of "edges" for converting data in one format into another format. Each edge corresponds to a conversion routine for converting data from one format to another. Each edge is part of a "protocol" (or more generally a component) that may include multiple related edges. For example, a protocol may have edges that each convert data in one format into several different formats. Each edge has an input format and an output format. The label map get routine identifies a sequence of edges such that the output format of each edge is compatible with the input format of another edge in the sequence, except for the input format of the first edge in the sequence and the output format of the last edge in the sequence. FIG. **2** is a block diagram illustrating a sequence of edges. Protocol PI includes an edge for converting format D**1** to format D**2** and an edge for converting format D**1** to format D**3**; protocol P**2** includes an edge for converting format D**2** to format D**5**, and so on. A 30 sequence for converting format D **1** to format D **15** is shown by the curved lines and is defined by the address "P**1**:I, P**2**:**1**, P**3**:**2**, P**4**:**7**." When a packet of data in format D I is processed by this sequence, it is converted to format DIS. During the process, the packet of data is sequentially converted to format D**2**, D**5**, and D**13**. The output format of protocol P**2**, edge **1** (i.e., P**2**: **1**) is format D**5**, but the input format of P**3**:**2** is format D**10**. The label map get routine uses an aliasing mechanism by which two formats, such as D**5** and D**10** are identified as being compatible. The use of aliasing allows different names of the same format or compatible formats to be correlated.

FIG. **3** is a block diagram illustrating components of the conversion system in one embodiment. The conversion system **300** can operate on a computer system with a central processing unit **301**, I/O devices **302**, and memory **303**. The 110 devices may include an Internet connection, a connection to various output devices such as a television, and a connection to various input devices such as a television receiver. The media mapping system may be stored as instructions on a computer-readable medium, such as a disk drive, memory, or data transmission medium. The data structures of the media mapping system may also be stored on a computer-readable medium. The conversion system includes drivers **304**, a• forwarding component **305**, a demux component **306**, a label map get component **307**, path data structures **308**, conversion routines **309**, and instance data **310**. Each driver receives data in a source format and forwards the data to the forwarding component. The forwarding component identifies the next conversion routine in the path and invokes that conversion routine to process a packet. The forwarding component may invoke the demux component to search for the next conversion routine and add that conversion routine to the path. The demux component may invoke the label map get component to identify the next conversion routine to process the packet. The demux component stores information defining the paths in the path structures. The conversion routines store their state information in the instance data.

FIG. **4** is a block diagram illustrating example path data structures in one embodiment. The demux component identifies a sequence of "edges" for converting data in one format into another format by invoking the label map get component. Each edge corresponds to a conversion routine for converting data from one format to another. As discussed above, each edge is part of a "protocol" that may include multiple related edges. For example, a protocol may have edges that each convert data in one format into several different formats. Each edge has as an input format ("input label") and an output format ("output label"). Each rectangle represents a session **410**, **420**, **430**, **440**, **450** for a protocol. A session corresponds to an instance of a protocol. That is, the session includes the protocol and state information associated with that instance of the protocol. Session **410** corresponds to a session for an Ethernet protocol; session **420** corresponds to a session for an IP protocol; and sessions **430**, **440**, **450** correspond to sessions for a TCP protocol. FIG. **4** illustrates three paths **461**, **462**, **463**. Each path includes edges **411**, **421**, **431**. The paths share the same Ethernet session **410** and IP session **420**, but each path has a unique TCP session **430**, **440**, **450**. Thus, path **461** includes sessions **410**, **420**, and **430**; path **462** includes sessions **410**, **420**, and **440**; and path **463** includes sessions **410**, **420**, and **450**. The conversion system represents each path by a sequence of path entry structures. Each path entry structure is represented by a triangle. Thus, path **461** is represented by path entries **415**, **425**, and **433**. The conversion system represents the path entries of a path by a stack list. Each path also has a queue **471**, **472**, **473** associated with it. Each queue stores the messages that are to be processed by the conversion routines of the edges of the path. Each session includes a binding **412**, **422**, **432**, **442**, **452** that is represented by an oblong shape adjacent to the corresponding edge. A binding for an edge of a session represents those paths that include the edge. The binding **412** indicates that three paths are bound (or "nailed") to edge **411** of the Ethernet session **410**. The conversion system uses a path list to track the paths that are bound to a binding. The path list of binding **412** identifies path entries **413**, **414**, and **415**.

FIG. **5** is a block diagram that illustrates the interrelationship of the data structures of a path. Each path has a corresponding path structure **501** that contains status information and pointers to a message queue structure **502**, a stack list structure **503**, and a path address structure **504**. The status of a path can be extend, continue, or end. Each message handler returns a status for the path. The status of extend means that additional path entries should be added to the path. The status of end means that this path should end at this point and subsequent processing should continue at a new path. The status of continue means that the protocol does not care how the path is handled. In one embodiment, when a path has a status of continue, the system creates a copy of the path and extends the copy. The message queue structure identifies the messages (or packets of a message) that are queued up for processing by the path and identifies the path entry at where the processing should start. The stack list structure contains a list of pointers to the path entry structures **505** that comprise the path. Each path entry structure contains a pointer to the corresponding path data structure, a pointer to a map structure **507**, a pointer to a multiplex list **508**, a pointer to the corresponding path address structure, and a pointer to a member structure **509**. A map structure identifies the output label of the edge of the path entry and optionally a target label and a target key. A target key identifies the session associated with the protocol that converts the packet to the target label. (The terms "media," "label," and "format" are used interchangeably to refer to the output of a protocol.) The multiplex list is used during the demux process to track possible next edges when a path is being identified as having more than one next edge. The member structure indicates that the path entry represents an edge of a path and contains a pointer to a binding structure to which the path entry is associated (or "nailed"), a stack list entry is the position of the path entry within the associated stack list, a path list entry is the position of the path entry within the associated path list of a binding and an address entry is the position of the binding within the associated path address. A path address of a path identifies the bindings to which the path entries are bound. The path address structure contains a URL for the path, the name of the path identified by the address, a pointer to a binding list structure **506**, and the identification of the current binding within the binding list. The URL (e.g., "protocol://tcp(0)/ip(0)/eth(0)") identifies conversion routines (e.g., protocols and edges) of a path in a human-readable format. The URL (universal resource locator) includes a type field (e.g., "protocol") followed by a sequence of items (e.g., "tcp(0)"). The type field specifies the format of the following information in the URL, that specifies that the type field is followed by a sequence of items. Each item identifies a protocol and an edge (e.g., the protocol is "tcp" and the edge is "0"). In one embodiment, the items of a URL may also contain an identifier of state information that is to be used when processing a message. These URLs can be used to illustrate to a user various paths that are available for processing a message. The current binding is the last binding in the path as the path is being built. The binding list structure contains a list of pointers to the binding structures associated with the path. Each binding structure **510** contains a pointer to a session structure, a pointer to an edge structure, a key, a path list structure, and a list of active paths through the binding. The key identifies the state information for a session of a protocol. A path list structure contains pointers to the path entry structures associated with the binding.

FIG. **6** is a block diagram that illustrates the interrelationship of the data structures associated with a session. A session structure **601** contains the context for the session, a pointer to a protocol structure for the session, a pointer to a binding table structure **602** for the bindings associated with the session, and the key. The binding table structure contains a list of pointers to the binding structures **510** for the session. The binding structure is described above with reference to FIG. **5**. The path list structure **603** of the binding structure contains a list of pointers to path entry structures **505**. The path entry structures are described with reference to FIG. **5**.

FIGS. **7** A, 7B, and **7**C comprise a flow diagram illustrating the processing of the message send routine. The message send routine is passed a message along with the path entry associated with the session that last processed the message. The message send routine invokes the message handler of the next edge in the path or queues the message for processing by a path. The message handler invokes the demux routine to identify the next path entry of the path. When a driver receives a message, it invokes the message send routine passing a reference path entry. The message send routine examines the passed path entry to determine (1) whether multiple paths branch from the path of the passed path entry, (2) whether the passed path entry is a reference with an associated path, or (3) whether the passed path entry is a member with a next path entry. If multiple paths branch from the path of the passed path entry, then the routine recursively invokes the message send routine for each path. If the path entry is a reference with an associated path, then the driver previously invoked the message send routine, which associated a path with the reference path entry, and the routine places the message on the queue for the path. If the passed path entry is a member with a next path entry, then the routine invokes the message handler (i.e., conversion routine of the edge) associated with the next path entry. If the passed path entry is a reference without an associated path or is a member without a next path entry, then the routine invokes the demux routine to identify the next path entry. The routine then recursively invokes the messages send routine passing that next path entry. In decision block **701**, if the passed path entry has a multiplex list, then the path branches off into multiple paths and the routine continues at block **709**, else the routine continues at block **702**. A packet may be processed by several different paths. For example, if a certain message is directed to two different output devices, then the message is processed by two different paths. Also, a message may need to be processed by multiple partial paths when searching for a complete path. In decision block **702**, if the passed path entry is a member, then either the next path entry indicates a nailed binding or the path needs to be extended and the routine continues at block **704**, else the routine continues at block **703**. A nailed binding is a binding (e.g., edge and protocol) is associated with a session. In decision block **703**, the passed path entry is a reference and if the passed path entry has an associated path, then the routine can queue the message for the associated path and the routine continues at block **703**A, else the routine needs to identify a path and the routine continues at block **707**. In block **703**A, the routine sets the entry to the first path entry in the path and continues at block **717**. In block **704**, the routine sets the variable position to the stack list entry of the passed path entry. In decision block **705**, the routine sets the variable next entry to the next path entry in the path. If there is a next entry in the path, then the next session and edge of the protocol have been identified and the routine continues at block **706**, else the routine continues at block **707**. In block **706**, the routine passes the message to the message handler of the edge associated with the next entry and then returns. In block **706**, the routine invokes the demux routine passing the passed message, the address of the passed path entry, and the passed path entry. The demux routine returns a list of candidate paths for processing of the message. In decision block **708**, if at least one candidate path is returned, then the routine continues at block **709**, else the routine returns.

Blocks **709**-**716** illustrate the processing of a list of candidate paths that extend from the passed path entry. In blocks **710**-**716**, the routine loops selecting each candidate path and sending the message to be process by each candidate path. In block **710**, the routine sets the next entry to the first path entry of the next candidate path. In decision block **711**, if all the candidate paths have not yet been processed, then the routine continues at block **712**, else the routine returns. In decision block **712**, if the next entry is equal to the passed path entry, then the path is to be extended and the routine continues at block **705**, else the routine continues at block **713**. The candidate paths include a first path entry that is a reference path entry for new paths or that is the last path entry of a path being extended. In decision block **713**, if the number of candidate paths is greater than one, then the routine continues at block **714**, else the routine continues at block **718**. In decision block **714**, if the passed path entry has a multiplex list associated with it, then the routine continues at block **716**, else the routine continues at block **715**. In block **715**, **11** the routine associates the list of candidate path with the multiplex list of the passed path entry and continues at block **716**. In block **716**, the routine sends the message to the next entry by recursively invoking the message send routine. The routine then loops to block **710** to select the next entry associated with the next candidate path.

Blocks **717**-**718** are performed when the passed path entry is a reference path entry that has a path associated with it. In block **717**, if there is a path associated with the next entry, then the routine continues at block **718**, else the routine returns. In block **718**, the routine queues the message for the path of the next entry and then returns.

FIG. **8** is a flow diagram of the demux routine. This routine is passed the packet (message) that is received, an address structure, and a path entry structure. The demux routine extends a path, creating one if necessary. The routine loops identifying the next binding (edge and protocol) that is to process the message and "nailing" the binding to a session for the message, if not already nailed. After identifying the nailed binding, the routine searches for the shortest path through the nailed binding, creating a path if none exists. In block **801**, the routine invokes the initialize demux routine. In blocks **802**-**810**, the routine loops identifying a path or portion of a path for processing the passed message. In decision block **802**, if there is a current status, which was returned by the demuxkey routine that was last invoked (e.g., continue, extend, end, or postpone), then the routine continues at block **803**, else the routine continues at block **811**. In block **803**, the routine invokes the get next binding routine. The get next binding routine returns the next binding in the path. The binding is the edge of a protocol. That routine extends the path as appropriate to include the binding. The routine returns a return status of break, binding, or multiple. The return status of binding indicates that the next binding in the path was found by extending the path as appropriate and the routine continues to "nail" the binding to a session as appropriate. The return status of multiple means that multiple trails (e.g., candidate paths) were identified as possible extensions of the path. In a decision block **804**, if the return status is break, then the routine continues at block **811**. If the return status is multiple, then the routine returns. If the return status is binding, then the routine continues at block **805**. In decision block **805**, if the retrieved binding is nailed as indicated by being assigned to a session, then the routine loops to block **802**, else the routine continues at block **806**. In block **806**, the routine invokes the get key routine of the edge associated with the binding. The get key routine creates the key for the session associated with the message. If a key cannot be created until subsequent bindings are processed or because the current binding is to be removed, then the get key routine returns a next binding status, else it returns a continue status. In decision block **807**, if the return status of the get key routine is next binding, then the routine loops to block **802** to get the next binding, else the routine continues at block **808**. In block **808**, the routine invokes the routine get session. The routine get session returns the session associated with the key, creating a new session if necessary. In block **809**, the routine invokes the routine nail binding. The routine nail binding retrieves the binding if one is already nailed to the session. Otherwise, that routine nails the binding to the session. In decision block **810**, if the nail binding routine returns a status of simplex, then the routine continues at block **811** because only one path can use the session, else the routine loops to block **802**. Immediately upon return from the nail binding routine, the routine may invoke a set map routine of the edge passing the session and a map to allow the edge to set its map. In block **811**, the routine invokes the find path routine, which finds the shortest path through the binding list and creates a path if necessary. In block **812**, the routine invokes the process path hopping routine, which determines whether the identified path is part of a different path. Path hopping occurs when, for example, IP fragments are built up along separate paths, but once the fragments are built up they can be processed by the same subsequent path.

FIG. **9** is a flow diagram of the initialize demux routine. This routine is invoked to initialize the local data structures that are used in the demux process and to identify the initial binding. The demux routine finds the shortest path from the initial binding to the final binding. If the current status is demux extend, then the routine is to extend the path of the passed path entry by adding additional path entries. If the current status is demux end, then the demux routine is ending the current path. If the current status is demux continue, then the demux routine is in the process of continuing to extend or in the process of starting a path identified by the passed address. In block **901**, the routine sets the local map structure to the map structure in the passed path entry structure. The map structure identifies the output label, the target label, and the target key. In the block **902**, the routine initializes the local message structure to the passed message structure and initializes the pointers path and address element to null. In block **903**, the routine sets of the variable saved status to 0 and the variable status to demux continue. The variable saved status is used to track the status of the demux process when backtracking to nail a binding whose nail was postponed. In decision block **904**, if the passed path entry is associated with a path, then the routine continues at block **905**, else the routine continues at block **906**. In block **905**, the routine sets the variable status to the status of that path. In block **906**, if the variable status is demux continue, then the routine continues at block **907**. If the variable status is demux end, then the routine continues at block **908**. If the variable status is demux extend, then the routine continues at block **909**. In block **907**, the status is demux continue, and the routine sets the local pointer path address to the passed address and continues at block **911**. In block **908**, the status is demux end, and the routine invokes the init end routine and continues at block **911**. In block **909**, the status is demux extend, and the routine sets the local path address to the address of the path that contains the passed path entry. In block **910**, the routine sets the address element and the current binding of the path address pointed to by the local pointer path address to the address entry of the member structure of the passed path entry. In the block **911**, the routine sets the local variable status to demux continue and sets the local binding list structure to the binding list structure from the local path address structure. In block **912**, the routine sets the local pointer current binding to the address of the current binding pointed to by local pointer path address and sets the local variable postpone to 0. In block **913**, the routine sets the function traverse to the function that retrieves the next data in a list and sets the local pointer session to null. The routine then returns.

FIG. **10** is a flow diagram of the init end routine. If the path is simplex, then the routine creates a new path from where the other one ended, else the routine creates a copy of the path. In block **1001**, if the binding of the passed path entry is simplex (i.e., only one path can be bound to this binding), then the routine continues at block **1002**, else the routine continues at block **1003**. In block **1002**, the routine sets the local pointer path address to point to an address structure that is a copy of the address structure associated with the passed path entry structure with its current binding to the address entry associated with the passed path entry structure, and then returns. In block **1003**, the routine sets the local pointer path address to point to an address structure that contains the URL of the path that contains the passed path entry. In block **1004**, the routine sets the local pointer element to null to initialize the selection of the bindings. In blocks **1005** through **1007**, the routine loops adding all the bindings for the address of the passed path entry that include and are before the passed path entry to the address pointed to by the local path address. In block **1005**, the routine retrieves the next binding from the binding list starting with the first. If there is no such binding, then the routine returns, else the routine continues at block **1006**. In block **1006**, the routine adds the binding to the binding list of the local path address structure and sets the current binding of the local variable path address. In the block **1007**, if the local pointer element is equal to the address entry of the passed path entry, then the routine returns, else the routine loops to block **1005** to select the next binding.

FIG. **11** is a flow diagram of a routine to get the next binding. This routine returns the next binding from the local binding list. If there is no next binding, then the routine invokes the routine label map get to identify the list of edges ("trails") that will map the output label to the target label. If only one trail is identified, then the binding list of path address is extended by the edges of the trail. If multiple trails are identified, then a path is created for each trail and the routine returns so that the demux process can be invoked for each created path. In block **1101**, the routine sets the local pointer binding to point to the next or previous (as indicated by the traverse function) binding in the local binding list. In block **1102**, if a binding was found, then the routine returns an indication that a binding was found, else the routine continues at block **1103**. In block **1103**, the routine invokes the label map get function passing the output label and target label of the local map structure. The label map get function returns a trail list. A trail is a list of edges from the output label to the target label. In decision block **1104**, if the size of the trail list is one, then the routine continues at block **1105**, else the routine continues at block **1112**. In blocks **1105-1111**, the routine extends the binding list by adding a binding data structure for each edge in the trail. The routine then sets the local binding to the last binding in the binding list. In block **1108**, the routine sets the local pointer current binding to point to the last binding in the local binding list. In block **1106**, the routine sets the local variable temp trail to the trail in the trail list. In block **1107**, the routine extends the binding list by temp trail by adding a binding for each edge in the trail. These bindings are not yet nailed. In block **1108**, the routine sets the local binding to point to the last binding in the local binding list. In decision block **1109**, if the local binding does not have a key for a session and the local map has a target key for a session, then the routine sets the key for the binding to the target key of the local map and continues at block **1110**, else the routine loops to block **1101** to retrieve the next binding in path. In block **1110**, the routine sets the key of the local binding to the target key of the local map. In block **1111**, the routine sets the target key of the local map to null and then loop to block **1101** to return the next binding. In decision block **1112**, if the local session is set, then the demultiplexing is already in progress and the routine returns a break status. In block **1113**, the routine invokes a prepare multicast paths routine to prepare a path entry for each trail in the trail list. The routine then returns a multiple status.

FIG. **12** is a flow diagram of the get key routine. The get key routine invokes an edge's demuxkey routine to retrieve a key for the session associated with the message. The key identifies the session of a protocol. The demux key routine creates the appropriate key for the message. The demux key routine returns a status of remove, postpone, or other. The status of remove indicates that the current binding should be removed from the path. The status of postpone indicates that the demux key routine cannot create the key because it needs information provided by subsequent protocols in the path. For example, a TCP session is defined by a combination of a remote and local port address and an IP address. Thus, the TCP protocol postpones the creating of a key until the IP protocol identifies the IP address. The get key routine returns a next binding status to continue at the next binding in the path. Otherwise, the routine returns a continue status. In block **1201**, the routine sets the local edge to the edge of the local binding (current binding) and sets the local protocol to the protocol of the local edge. In block **1202**, the routine invokes the demux key routine of the local edge passing the local message, local path address, and local map. The demux key routine sets the key in the local binding. In decision block **1203**, if the demux key routine returns a status of remove, then the routine continues at block **1204**. If the demux key routine returns a status of postpone, then the routine continues at block **1205**, else the routine continues at block **1206**. In block **1204**, the routine sets the flag of the local binding to indicate that the binding is to be removed and continues at block **1206**. In block **1205**, the routine sets the variable traverse to the function to list the next data, increments the variable postpone, and then returns a next

binding status. In blocks **1206-1214**, the routine processes the postponing of the creating of a key. In blocks **1207-1210**, if the creating of a key has been postponed, then the routine indicates to backtrack on the path, save the demux status, and set the demux status to demux continue. In blocks **1211-1213**, if the creating of a key has not been postponed, then the routine indicates to continue forward in the path and to restore any saved demux status. The save demux status is the status associated by the binding where the backtrack started. In decision block **1206**, if the variable postpone is set, then the routine continues at block **1207**, else the routine continues at block **1211**. In block **1207**, the routine decrements the variable postpone and sets the variable traverse to the list previous data function. In decision block **1208**, if the variable saved status is set, then the routine continues at block **1210**, else the routine continues at block **1209**. The variable saved status contains the status of the demux process when the demux process started to backtrack. In block **1209**, the routine sets the variable saved status to the variable status. In block **1210**, the routine sets the variable status to demux continue and continues at block **1214**. In block **1211**, the routine sets the variable traverse to the list next data function. In decision block **1212**, if the variable saved status in set, then the routine continues at block **1213**, else the routine continues at block **1214**. In block **1213**, the routine sets the variable status to the variable saved status and sets the variable saved status to 0. In decision block **1214**, if the local binding indicates that it is to be removed, then the routine returns a next binding status, else the routine returns a continue status.

FIG. **13** is a flow diagram of the get session routine. This routine retrieves the session data structure, creating a data structure session if necessary, for the key indicated by the binding. In block **1301**, the routine retrieves the session from the session table of the local protocol indicated by the key of the local binding. Each protocol maintains a mapping from each key to the session associated with the key. In decision block **1302**, if there is no session, then the routine continues at block **1303**, else the routine returns. In block **1303**, the routine creates a session for the local protocol. In block **1304**, the routine initializes the key for the local session based on the key of the local binding. In block **1305**, the routine puts the session into the session table of the local protocol. In block **1306**, the routine invokes the create session function of the protocol to allow the protocol to initialize its context and then returns.

FIG. **14** is a flow diagram of the nail binding routine. This routine determines whether a binding is already associated with ("nailed to") the session. If so, the routine returns that binding. If not, the routine associates the binding with the session. The routine returns a status of simplex to indicate that only one path can extend through the nailed binding. In decision block **1401**, if the binding table of the session contains an entry for the edge, then the routine continues at block **1402**, else the routine continues at block **1405**. In block **1402**, the routine sets the binding to the entry from the binding table of the local session for the edge. In block **1403**, the routine sets the current binding to point to the binding from the session. In block **1404**, if the binding is simplex, then the routine returns a simplex status, else the routine returns. Blocks **1405** through **1410** are performed when there is no binding in the session for the edge. In block **1405**, the routine sets the session of the binding to the variable session. In block **1406**, the routine sets the key of the binding to the key from the session. In block **1407**, the routine sets the entry for the edge in the binding table of the local session to the binding. In block **1408**, the routine invokes the create binding function of the edge of the binding passing the binding so the edge can initialize the binding. If that function returns a status of remove, the routine continues at block **1409**. In block **1409**, the routine sets the binding to be removed and then returns.

FIG. **15** is a flow diagram of the find path routine. The find path routine identifies the shortest path through the binding list. If no such path exists, then the routine extends a path to include the binding list. In decision block **1501**, if the binding is simplex and a path already goes through this binding (returned as an entry), then the routine continues at block **1502**, else the routine continues at block **1503**. In block **1502**, the routine sets the path to the path of the entry and returns. In block **1503**, the routine initializes the pointers element and short entry to null. In block **1504**, the routine sets the path to the path of the passed path entry. If the local path is not null and its status is demux extend, then the routine continues at block **1509**, else the routine continues at block **1505**. In blocks **1505-1508**, the routine loops identifying the shortest path through the bindings in the binding list. The routine loops selecting each path through the binding. The selected path is eligible if it starts at the first binding in the binding list and the path ends at the binding. The routine loops setting the short entry to the shortest eligible path found so far. In block **1505**, the routine sets the variable first binding to the first binding in the binding list of the path address. In block **1506**, the routine selects the next path (entry) in the path list of the binding starting with the first. If a path is selected (indicating that there are more paths in the binding), then the routine continues at block **1507**, else the routine continues at block **1509**. In block **1507**, the routine determines whether the selected path starts at the first binding in the binding list, whether the selected path ends at the last binding in the binding list, and whether the number of path entries in the selected path is less than the number of path entries in the shortest path selected so far. If these conditions are all satisfied, then the routine continues at block **1508**, else the routine loops to block **1506** to select the next path (entry). In block **1508**, the routine sets the shortest path (short entry) to the selected path and loops to block **1506** to select the next path through the binding. In block **1509**, the routine sets the selected path (entry) to the shortest path. In decision block **1510**, if a path has been found, then the routine continues at block **1511**, else the routine continues at block **1512**. In block **1511**, the routine sets the path to the path of the selected path entry and returns. Blocks **1512-1516** are performed when no paths have been found. In block **1512**, the routine sets the path to the path of the passed path entry. If the passed path entry has a path and its status is demux extend, then the routine continues at block **1515**, else the routine continues at block **1513**. In block **1513**, the routine creates a path for the path address. In block **1514**, the routine sets the variable element to null and sets the path entry to the first element in the stack list of the path. In block **1515**, the routine sets the variable element to be address entry of the member of the passed path entry and sets the path entry to the passed path entry. In block **1516**, the routine invokes the extend path routine to extend the path and then returns. The extend path routine creates a path through the bindings of the binding list and sets the path status to the current demux status.

FIG. **16** is a flow diagram of the process of path hopping routine. Path hopping occurs when the path through the binding list is not the same path as that of the passed path entry. In decision block **1601**, if the path of the passed path entry is set, then the routine continues at block **1602**, else the routine continues at block **1609**. In decision block **1602**, if the path of the passed path entry is equal to the local path, then the routine continues at **1612**, else path hopping is occurring and the routine continues at block **1603**. In blocks **1603-1607**, the routine loops positioning pointers at the first path entries of the paths that are not at the same binding. In block **1603**, the routine sets the variable old stack to the stack list of the path of the passed path entry. In block **1604**, the routine sets the variable new stack to the stack list of the local path. In block **1605**, the routine sets the variable old element to the next element in the old stack. In block **1606**, the routine sets the variable element to the next element in the new stack. In decision block **1607**, the routine loops until the path entry that is not in the same binding is located. In decision block **1608**, if the variable old entry is set, then the routine is not at the end of the hopped from path and the routine continues at block **1609**, else routine continues at block **1612**. In block **1609**, the routine sets the variable entry to the previous entry in the hopped-to path. In block **1610**, the routine sets the path of the passed path entry to the local path. In block **1611**, the routine sets the local entry to the first path entry of the stack list of the local path. In block **1612**, the routine inserts an entry into return list and then returns.

Although the conversion system has been described in terms of various embodiments, the invention is not limited to these embodiments. Modification within the spirit of the invention will be apparent to those skilled in the art. For example, a conversion routine may be used for routing a message and may perform no conversion of the message. Also, a reference to a single copy of the message can be passed to each conversion routine or demuxkey routine. These routines can advance the reference past the header information for the protocol so that the reference is positioned at the next header. After the demux process, the reference can be reset to point to the first header for processing by the conversion routines in sequence. The scope of the invention is defined by the claims that follow.

What is claimed is:

1. An apparatus, comprising:
- a processing unit; and
- a memory storing instructions executable by the processing unit to:
  - receive one or more packets of a message;
  - determine a key value using information in the one or more packets;
  - identify, using the key value, a sequence of two or more routines, wherein the sequence includes a routine that is used to execute a Transmission Control Protocol (TCP) to process packets having a TCP format;
  - create a path that includes one or more data structures that indicate the identified sequence of two or more routines, wherein the path is usable to store state information associated with the message; and
  - process subsequent packets in the message using the sequence of two or more routines indicated in the path.

2. The apparatus of claim **1**, wherein the key value includes an IP address and one or more TCP port addresses.

3. The apparatus of claim **1**, wherein the sequence of two or more routines includes:
- a second routine that is used to execute a second protocol to process packets having a format other than the TCP format, wherein the second protocol is an application-level protocol.

4. The apparatus of claim **3**, wherein the second protocol is an HTTP protocol.

5. The apparatus of claim **1**, wherein the path further indicates sessions corresponding to respective ones of the sequence of two or more routines.

6. The apparatus of claim **1**, wherein the path indicates the key value and a queue for storing packets associated with the message.

7. The apparatus of claim **1**, wherein the sequence of two or more routines includes a routine that is executable to process information in the message without performing a format conversion.

8. The apparatus of claim **1**, wherein the memory stores instructions executable by the processing unit to:

store, prior to receiving any packets of the message, a list having a plurality of sequences of routines that includes the sequence of two or more routines.

9. The apparatus of claim **8**, wherein the instructions executable by the processing unit to identify the sequence of two or more routines are executable to select the sequence of two or more routines from the list using the key value.

10. A non-transitory, computer-readable medium comprising software instructions for processing a message, wherein the software instructions, when executed, cause a computer system to:

determine a key value based on information in one or more received packets of the message, wherein the key value includes an IP address and one or more Transmission Control Protocol (TCP) port addresses;

identify, using the key value, a sequence of two or more routines;

create one or more data structures that reference the identified sequence of two or more routines and are usable to store state information associated with the message; and

process subsequent packets in the message using the identified sequence of two or more routines, wherein the sequence includes a routine that is used to execute TCP to process at least one of the subsequent packets having a TCP format.

11. The computer-readable medium of claim **10**, wherein the created one or more data structures include a reference to the key value and to a queue for storing packets of the message for processing.

12. The computer-readable medium of claim **10**, wherein the one or more data structures indicate sessions corresponding to respective ones of the sequence of two or more routines.

13. The computer-readable medium of claim **10**, wherein the sequence of two or more routines includes a plurality of application-level routines.

14. The computer-readable medium of claim **10**, wherein the software instructions, when executed, further cause the computer system to store, prior to receiving any packets of the message, a list of routine sequences that includes the sequence of two or more routines.

15. The computer-readable medium of claim **14**, wherein the software instructions are executable to identify the sequence of two or more routines by selecting, using the key value, the sequence of two or more routines from the list.

16. A method, comprising:

receiving, by a computer network device, one or more packets of a message;

determining, by the computer network device, a key value based on information in the one or more packets;

identifying, by the computer network device, a particular sequence of two or more routines using the key value, wherein the particular sequence includes a routine that is used to execute a Transmission Control Protocol (TCP) to process packets having a TCP format;

creating, by the computer network device, one or more data structures that reference the particular sequence, wherein the one or more data structures are usable to store state information associated with the message; and

processing, by the computer network device, subsequent packets in the message using the particular sequence.

17. The method of claim **16**, further comprising:

storing, by the computer network device prior to receiving any packets of the message, a list that indicates a plurality of sequences of routines that includes the particular sequence.

18. The method of claim **17**, wherein the identifying includes using the key value to select the particular sequence from the list.

19. The method of claim **16**, wherein the key value includes an IP address and one or more TCP port addresses.

20. The method of claim **16**, wherein the creating includes storing, in the one or more data structures:

an indication of the key value; and

a reference to a queue for storing packets of the message.

* * * * *

EXHIBIT 7

US010027780B2

(12) **United States Patent**
Balassanian

(10) **Patent No.: US 10,027,780 B2**
(45) **Date of Patent: *Jul. 17, 2018**

(54) **METHOD AND SYSTEM FOR DATA DEMULTIPLEXING**

(71) Applicant: **Implicit, LLC**, Seattle, WA (US)

(72) Inventor: **Edward Balassanian**, Seattle, WA (US)

(73) Assignee: **Implicit, LLC**, Seattle, WA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **15/703,964**

(22) Filed: **Sep. 13, 2017**

(65) **Prior Publication Data**

US 2018/0013865 A1 Jan. 11, 2018

Related U.S. Application Data

(63) Continuation of application No. 15/450,790, filed on Mar. 6, 2017, which is a continuation of application
(Continued)

(51) **Int. Cl.**
H04L 12/58 (2006.01)
H04L 29/06 (2006.01)
(Continued)

(52) **U.S. Cl.**
CPC ***H04L 69/08*** (2013.01); ***H04L 29/06*** (2013.01); ***H04L 45/00*** (2013.01);
(Continued)

(58) **Field of Classification Search**
None
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,298,674 A 3/1994 Yun
5,392,390 A 2/1995 Crozier
(Continued)

FOREIGN PATENT DOCUMENTS

EP 0408132 1/1991
EP 0807347 11/1997
(Continued)

OTHER PUBLICATIONS

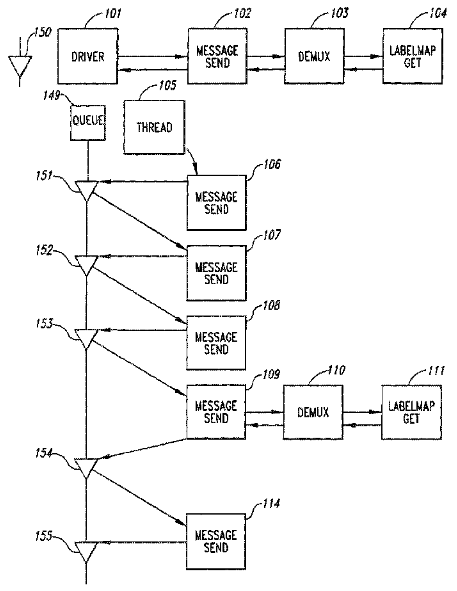
Non-Final Office Action in Inter Partes Reexamination Control No. 95/000,659 dated Aug. 16, 2013, 107 pages.
(Continued)

Primary Examiner — Duc T Duong
(74) *Attorney, Agent, or Firm* — Meyertons, Hood, Kivlin, Kowert & Goetzel, P.C.

(57) **ABSTRACT**

A method and system for demultiplexing packets of a message is provided.
The demultiplexing system receives packets of a message, identifies a sequence of message handlers for processing the message, identifies state information associated with the message for each message handler, and invokes the message handlers passing the message and the associated state information. The system identifies the message handlers based on the initial data type of the message and a target data type. The identified message handlers effect the conversion of the data to the target data type through various intermediate data types.

20 Claims, 16 Drawing Sheets



Related U.S. Application Data

No. 15/050,027, filed on Feb. 22, 2016, now Pat. No. 9,591,104, which is a continuation of application No. 14/230,952, filed on Mar. 31, 2014, now Pat. No. 9,270,790, which is a continuation of application No. 13/911,324, filed on Jun. 6, 2013, now Pat. No. 8,694,683, which is a continuation of application No. 13/236,090, filed on Sep. 19, 2011, now abandoned, which is a continuation of application No. 10/636, 314, filed on Aug. 6, 2003, now Pat. No. 8,055,786, which is a continuation of application No. 09/474, 664, filed on Dec. 29, 1999, now Pat. No. 8,629,163.

(51) **Int. Cl.**

H04L 29/08	(2006.01)
H04L 29/12	(2006.01)
H04L 12/701	(2013.01)

(52) **U.S. Cl.**
CPC ***H04L 61/2007*** (2013.01); ***H04L 61/6063*** (2013.01); ***H04L 67/02*** (2013.01); ***H04L 69/16*** (2013.01); ***H04L 69/18*** (2013.01); ***H04L 69/22*** (2013.01); ***H04L 69/32*** (2013.01)

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,414,833	A	5/1995	Hershey et al.
5,425,029	A	6/1995	Hluchyj et al.
5,568,478	A	10/1996	van Loo, Jr. et al.
5,627,997	A	5/1997	Pearson et al.
5,710,917	A	1/1998	Musa et al.
5,727,159	A	3/1998	Kikinis
5,740,430	A	4/1998	Rosenberg et al.
5,761,651	A	6/1998	Hasebe
5,768,521	A	6/1998	Dedrick
5,826,027	A	10/1998	Pedersen et al.
5,835,726	A	11/1998	Shwed et al.
5,842,040	A	11/1998	Hughes et al.
5,848,233	A	12/1998	Radia et al.
5,848,246	A	12/1998	Gish
5,848,415	A	12/1998	Guck
5,854,899	A	12/1998	Callon et al.
5,870,479	A	2/1999	Feiken et al.
5,896,383	A	4/1999	Wakeland
5,898,830	A	4/1999	Wesinger, Jr. et al.
5,918,013	A	6/1999	Mighdoll et al.
5,983,348	A	11/1999	Ji
5,987,256	A	11/1999	Wu et al.
5,991,299	A	11/1999	Radogna et al.
5,991,806	A	11/1999	McHann, Jr.
6,032,150	A	2/2000	Nguyen
6,035,339	A	3/2000	Agraharam et al.
6,047,002	A	4/2000	Hartmann et al.
6,067,575	A	5/2000	McManis et al.
6,091,725	A	7/2000	Cheriton et al.
6,094,679	A	7/2000	Teng et al.
6,101,189	A	8/2000	Tsuruoka
6,101,320	A	8/2000	Schuetze et al.
6,104,500	A	8/2000	Alam et al.
6,104,704	A	8/2000	Buhler et al.
6,111,893	A	8/2000	Volftsun et al.
6,112,250	A	8/2000	Appelman
6,115,393	A	9/2000	Engel et al.
6,119,165	A	9/2000	Li et al.
6,119,236	A	9/2000	Shipley
6,122,666	A	9/2000	Beurket et al.
6,128,624	A	10/2000	Papiemiak et al.
6,130,917	A	10/2000	Monroe
6,141,749	A	10/2000	Coss et al.
6,151,390	A	11/2000	Volftsun et al.
6,157,622	A	12/2000	Tanaka et al.
6,167,441	A	12/2000	Himmel
6,192,419	B1	2/2001	Aditham et al.
6,199,054	B1	3/2001	Khan et al.
6,212,550	B1	4/2001	Segur
6,222,536	B1	4/2001	Kihl et al.
6,226,267	B1	5/2001	Spinney et al.
6,243,667	B1	6/2001	Kerr et al.
6,246,678	B1	6/2001	Erb et al.
6,259,781	B1	7/2001	Crouch et al.
6,275,507	B1	8/2001	Anderson et al.
6,278,532	B1	8/2001	Heimendinger et al.
6,292,827	B1	9/2001	Raz
6,356,529	B1	3/2002	Zarom
6,359,911	B1	3/2002	Movshovich et al.
6,374,305	B1	4/2002	Gupta et al.
6,401,132	B1	6/2002	Bellwood et al.
6,404,775	B1	6/2002	Leslie et al.
6,405,254	B1	6/2002	Hadland
6,426,943	B1	7/2002	Spinney et al.
6,427,171	B1 *	7/2002	Craft H04L 29/06 709/230
6,493,348	B1	12/2002	Gelman et al.
6,504,843	B1	1/2003	Cremin et al.
6,519,636	B2	2/2003	Engel et al.
6,560,236	B1	5/2003	Varghese et al.
6,574,610	B1	6/2003	Clayton et al.
6,578,084	B1	6/2003	Moberg et al.
6,598,034	B1	7/2003	Kloth
6,629,163	B1	9/2003	Balassanian
6,650,632	B1	11/2003	Volftsun et al.
6,651,099	B1	11/2003	Dietz et al.
6,678,518	B2	1/2004	Eerola
6,680,922	B1	1/2004	Jorgensen
6,701,432	B1	3/2004	Deng et al.
6,711,166	B1	3/2004	Amir et al.
6,772,413	B2	8/2004	Kuznetsov
6,785,730	B1	8/2004	Taylor
6,865,735	B1	3/2005	Sirer et al.
6,871,179	B1	3/2005	Kist et al.
6,889,181	B2	5/2005	Kerr et al.
6,937,574	B1	8/2005	Delaney et al.
6,957,346	B1	10/2005	Kivinen et al.
6,959,439	B1	10/2005	Boike
7,233,569	B1 *	6/2007	Swallow H04L 12/4633 370/225
7,233,948	B1	6/2007	Shamoon et al.
7,281,036	B1	10/2007	Lu et al.
7,383,341	B1	6/2008	Saito et al.
7,711,857	B2	5/2010	Balassanian
8,055,786	B2	11/2011	Balassanian
8,694,683	B2	4/2014	Balassanian
2003/0142669	A1	7/2003	Kubota et al.
2004/0015609	A1	1/2004	Brown et al.
2008/0250045	A1 *	10/2008	Balassanian G06F 17/30569
2009/0083763	A1	3/2009	Sareen et al.
2009/0265695	A1	10/2009	Karino
2017/0237668	A1 *	8/2017	Hall H04L 47/193 370/235

FOREIGN PATENT DOCUMENTS

EP	0817031	1/1998
JP	H10-49354	2/1998
JP	H10-55279	2/1998
JP	H10-74153	3/1998
JP	H10-289215	10/1998
WO	99/35799	7/1999

OTHER PUBLICATIONS

Decision on Petition in Reexamination Control No. 95/000,659 dated Aug. 19, 2013, 3 pages.
Response to Non-Final Office Action in Reexamination Control No. 95/000,659 dated Oct. 2, 2013 including Exhibits A-C, 37 pages.
Decision on Petition in Reexamination Control No. 95/000,660 dated Jul. 30, 2013, 12 pages.
Non-Final Office Action in Inter Partes Reexamination Control No. 95/000,660 dated Aug. 30, 2013, 23 pages.

(56) **References Cited**

OTHER PUBLICATIONS

RFC: 791. Internet Protocol: DARPA Internet Program Protocol Specification, Sep. 1981, prepared for Defense Advanced Research Projects Agency Information Processing Techniques Office by Information Sciences Institute University of Southern California, 52 pages.
2015 WL 2194627, United States District Court, N.D. California, *Implicit L.L.C.*, Plaintiff, v. *F5 Networks, Inc.*, Defendant, Case No. 14-cv-02856-SI, signed May 6, 2015, 14 pages.
Defendants' Invalidity Contentions Pursuant to Local Patent Rules 3-3 and 3-4, United States District Court Eastern District of Texas Tyler Division, *Implicit, LLC* v. *Trend Micro, Inc., Ericsson Inc., Huawei Technologies USA, Inc., NEC Corporation of America, Nokia Solutions and Networks US LLC*; Sep. 2, 2016, 53 pages.
Exhibits A-1-A16 Invalidity of U.S. Pat. No. 8,694,683, Defendants' Invalidity Contentions Pursuant to Local Patent Rules 3-3 and 3-4, United States District Court Eastern District of Texas Tyler Division, Sep. 2, 2016, 425 pages.
Exhibits B-1-B13 Invalidity of U.S. Pat. No. 9,270,790, Defendants' Invalidity Contentions Pursuant to Local Patent Rules 3-3 and 3-4, United States District Court Eastern District of Texas Tyler Division, Sep. 2, 2016, 369 pages.
Exhibits C-1-C21 Invalidity of U.S. Pat. No. 8,856,779, Defendants' Invalidity Contentions Pursuant to Local Patent Rules 3-3 and 3-4, United States District Court Eastern District of Texas Tyler Division, Sep. 2, 2016, 646 pages.
Exhibits D-1-D21 Invalidity of U.S. Pat. No. 9,325,740, Defendants' Invalidity Contentions Pursuant to Local Patent Rules 3-3 and 3-4, United States District Court Eastern District of Texas Tyler Division, Sep. 2, 2016, 419 pages.
Exhibits E-1-E20 Invalidity of U.S. Pat. No. 6,324,685, Defendants' Invalidity Contentions Pursuant to Local Patent Rules 3-3 and 3-4, United States District Court Eastern District of Texas Tyler Division, Sep. 2, 2016, 416 pages.
Michael Baentsch, et al., "WebMake: Integrating distributed software development in a structure-enhanced Web," Computer Networks and ISDN Systems 27 (1995), pp. 789-800.
Dan Decasper, et al., "A Scalable, High Performance Active Network Node," Apr. 1998, 21 pages.
John J. Hartman, et al., "Joust: A Platform for Liquid Software," Computer, IEEE, 1999, pp. 50-56.
David Mosberger, et al., "Making Paths Explicit in the Scout Operating System," Proceedings of the USENIX 2nd Symposium on Operating Systems Design and Implementation, Oct. 1996, 16 pages.
Oliver Spatscheck, et al., "Escort: A Path-Based OS Security Architecture," TR 97-17, Nov. 26, 1997, 17 pages.
Dan Decasper, et al., "DAN: Distrubuted Code Caching for Active Networks," IEEE, 1998, pp. 609-616.
Alexander, D. et al., "The SwitchWare Active Network Architecture", Jun. 6, 1998, IEEE.
Antoniazzi, S. et al., "An Open Software Architecture for Multimedia Consumer Terminals", Central Research Labs, Italy; Alcatel SEL Research Centre, Germany, ECMAST 1997.
Arbanowski, Stefan, "Generic Description of Telecommunication Services and Dynamic Resource Selection in Intelligent Communication Environments", Thesis, Technische Universitat Berlin, Oct. 9, 1996, (3 documents).
Arbanowski, S., et al., Service Personalization for Unified Messaging Systems, Jul. 6-8, 1999, The Fourth IEEE Symposium on Computers and Communications, ISCC '99, Red Sea, Egypt.
Atkinson, R., "Security Architecture for the Internet Protocol", Aug. 1995, Naval Research Laboratory.
Atkinson, R., "IP Authentication Header", Aug. 1995, Naval Research Laboratory.
Atkinson, R., "IP Encapsulating Security Payload (ESP)", Aug. 1995, Naval Research Laboratory.
Back, G., et al., Java Operating Systems: Design and Implementation, Aug. 1998, Technical Report UUCS-98-015, University of Utah.
Baker, Dr. Sean, "CORBA Implementation Issues", 1994, IONA Technologies, O'Reilly Institute Dublin, Ireland.
Barrett, R., et al., "Intermediaries: New Places for Producing and Manipulating Web Content", 1998, IBM Almaden Research Center, Elsevier Science.
Bellare, M., et al., "A Concrete Security Treatment of Symmetric Encryption: Analysis of the DES Modes of Operation", Aug. 15, 1997, Dept. of Computer Science and Engineering, University of California, San Diego.
Bellare, M., et al., "A Concrete Security Treatment of Symmetric Encryption: Analysis of the DES Modes of Operation", Aug. 15, 1997, IEEE.
Bellare, M., et al., "XOR MACs: New Methods for Message Authentication Using Finite Pseudorandom Functions", 1995, CRYPTO '95, LNCS 963, pp. 15-28, Springer-Verlag Berlin Heidelberg.
Bellissard, L., et al., "Dynamic Reconfiguration of Agent-Based Applications", Third European Research Seminar on Advances in Distributed Systems, (ERSADS '99) Madeira Island.
Bolding, Darren, "Network Security, Filters and Firewalls", 1995, www.acm.org/crossroads/xrds2-I/security.html.
Booch, G., et al., "Software Engineering with ADA", 1994, Third Edition, The Benjamin/Cummings Publishing Company, Inc., (2 documents).
Breugst, et al., "Mobile Agents—Enabling Technology for Active Intelligent Network Implementation", May/Jun. 1998, IEEE Network.
"C Library Functions", AUTH(3) Sep. 17, 1993, Solbourne Computer, Inc.
Chapman, D., et al., "Building Internet Firewalls", Sep. 1995, O'Reilly & Associates, Inc.
CheckPoint FireWall-1 Technical White Paper, Jul. 18, 1994, CheckPoint Software Technologies, Ltd.
CheckPoint FireWall-1 White Paper, Sep. 1995, Version 2.0, CheckPoint Software Technologies, Ltd.
Command Line Interface Guide P/N 093-0011-000 Rev C Version 2.5, 2000-2001, NetScreen Technologies, Inc.
Coulson, G. et al., "A CORBA Compliant Real-Time Multimedia Platform for Broadband Networks", Lecture Notes in Computer Science, 1996, Trends in Distributed Systems CORBA and Beyond.
Cox, Brad, "SuperDistribution, Objects As Property on the Electronic Frontier", 1996, Addison-Wesley Publishing Company.
Cranes, et al., "A Configurable Protocol Architecture for CORBA Environments", Autonomous Decentralized Systems 1997 Proceedings ISADS, Third International Symposium Apr. 9-11, 1997.
Curran, K., et al., "CORBA Lacks Venom", University of Ulster, Northern Ireland, UK 2000.
Dannert, Andreas, "Call Logic Service for a Personal Communication Supporting System", Thesis, Jan. 20, 1998, Technische Universitat Berlin.
DARPA Internet Program Protocol Specification, "Transmission Control Protocol", Sep. 1981, Information Sciences Institute, California.
DARPA Internet Program Protocol Specification, "Internet Protocol", Sep. 1981, Information Sciences Institute, California.
Decasper, D., et al., "Crossbow: A Toolkit for Integrated Services over Cell Switched IPv6", 1997, Computer Engineering and Networks Laboratory, ETH Zurich, Switzerland.
Decasper, D., et al., "Router Plugins a Software Architecture for Next Generation Routers", 1998, Proceedings of ACM SIGCONM '98.
Deering, S., et al., Internet Protocol, Version 6 (IPv6) Specification, Dec. 1998, Nokia, The Internet Society.
Deering, S., et al., Internet Protocol, Version 6 (IPv6) Specification, Dec. 1995, Network Working Group, RFC 1883.
Dutton, et al, "Asynchronous Transfer Mode Technical Overview (ATM)", Second Edition; IBM, Oct. 1995, 2nd Edition, Prentice Hall PTR, USA.
Eckardt, T., et al., "Application of X.500 and X.700 Standards for Supporting Personal Communications in Distributed Computing Environments", 1995, IEEE.

(56) **References Cited**

OTHER PUBLICATIONS

Eckardt, T., et al., "Personal Communications Support based on TMN and TINA Concepts", 1996, IEEE Intelligent Network Workshop (IN '96), Apr. 21-24, Melbourne, Australia.
Eckardt, T., et al., "Beyond IN and UPT—A Personal Communications Support System Based on TMN Concepts", Sep. 1997, IEEE Journal on Selected Areas in Communications, vol. 15, No. 7.
Egevang, K., et al., "The IP Network Address Translator (NAT)", May 1994, Network Working Group, RFC 1631.
Estrin, D., et al., "Visa Protocols for Controlling Inter-Organizational Datagram Flow", Dec. 1998, Computer Science Department, University of Southern California and Digital Equipment Corporation.
Faupel, M., "Java Distribution and Deployment", Oct. 9, 1997, APM Ltd., United Kingdom.
Felber, P., "The CORBA Object Group Service: A Service Approach to Object Groups in CORBA", Thesis, 1998, Ecole Polytechnique Federale de Lausanne, Switzerland.
Fish, R., et al., "DRoPS: Kernel Support for Runtime Adaptable Protocols", Aug. 25-27, 1998, IEEE 24th Euromicro Conference, Sweden.
Fiuczynski, M., et al., "An Extensible Protocol Architecture for Application-Specific Networking", 1996, Department of Computer Science and Engineering, University of Washington.
Franz, Stefan, "Job and Stream Control in Heterogeneous Hardware and Software Architectures", Apr. 1998, Technische Universitat, Berlin, (2 documents).
Fraser, T., "DTE Firewalls: Phase Two Measurement and Evaluation Report", Jul. 22, 1997, Trusted Information Systems, USA.
Gazis, V., et al., "A Survey of Dynamically Adaptable Protocol Stacks", first Quarter 2010, IEEE Communications Surveys & Tutorials, vol. 12, No. 1, 1st Quarter.
Gokhale, A., et al., "Evaluating the Performance of Demultiplexing Strategies for Real-Time CORBA", Nov. 1997, GLOBECOM.
Gokhale, A., et al., "Measuring and Optimizing CORBA Latency and Scalability Over High-Speed Networks", Apr. 1998, IEEE Transaction on Computers, vol. 47, No. 4; Proceedings of the International Conference on Distributed Computing Systems (ICDCS '97) May 27-30, 1997.
Gokhale, A., et al., "Operating System Support for High-Performance, Real-Time CORBA", 1996.
Gokhale, A., et al., "Principles for Optimizing CORBA Internet Inter-ORB Protocol Performance", Jan. 9, 1998, Proceedings of the HICSS Conference, Hawaii.
Gong, Li, "Java Security: Present and Near Future", May/Jun. 1997, IEEE Micro.
Gong, Li, "New Security Architectural Directions for Java (Extended Abstract)", Dec. 19, 1996, IEEE.
Gong, Li, "Secure Java Class Loading", Nov./Dec. 1998, IEEE Internet.
Goos, G., et al., "Lecture Notes in Computer Science: Mobile Agents and Security", 1998, Springer-Verlag Berlin Heidelberg.
Goralski, W., "Introduction to ATM Networking", 1995, McGraw-Hill Series on Computer Communications, USA.
Hamzeh, K., et al., "Layer Two Tunneling Protocol L2TP", Jan. 1998, PPP Working Group, Internet Draft.
Harrison, T., et al., "The Design and Performance of a Real-Time CORBA Event Service", Aug. 8, 1997,Proceedings of the OOPSLA '97 Conference, Atlanta, Georgia in Oct. 1997.
Huitema, Christian, "IPv6 the New Internet Protocol", 1997 Prentice Hall, Second Edition.
Hutchins, J., et al., "Enhanced Internet Firewall Design Using Stateful Filters Final Report", Aug. 1997, Sandia Report; Sandia National Laboratories.
IBM, Local Area Network Concepts and Products: Routers and Gateways, May 1996.
Juniper Networks Press Release, Juniper Networks Announces Junos, First Routing Operating System for High-Growth Internet Backbone Networks, Jul. 1, 1998, Juniper Networks.
Juniper Networks Press Release, Juniper Networks Ships the Industry's First Internet Backbone Router Delivering Unrivaled Scalability, Control and Performance, Sep. 16, 1998, Juniper Networks.
Karn, P., et al., "The ESP DES-CBC Transform", Aug. 1995, Network Working Group, RFC 1829.
Kelsey, J. et al., "Authenticating Outputs of Computer Software Using a Cryptographic Coprocessor", Sep. 1996, Cardis.
Krieger, D., et al., "The Emergence of Distributed Component Platforms", Mar. 1998, IEEE.
Krupczak, B., et al., "Implementing Communication Protocols in Java", Oct. 1998, IEEE Communications Magazine.
Krupczak, B., et al., "Implementing Protocols in Java: The Price of Portability", 1998, IEEE.
Lawson, Stephen, "Cisco NetFlow Switching Speeds Traffic Routing", Jul. 7, 1997, Infoworld.
Li, S., et al., "Active Gateway: A Facility for Video Conferencing Traffic Control", Feb. 1, 1997, Purdue University; Purdue e-Pubs; Computer Science Technical Reports.
Magedanz, T., et al., "Intelligent Agents: An Emerging Technology for Next Generation Telecommunications?", 1996, IEEE.
Mills, H., et al., "Principles of Information Systems Analysis and Design", 1986, Academic Press, Inc., (2 documents).
Mosberger, David, "*Scout: A Path-Based Operating System*", Doctoral Dissertation Submitted to the University of Arizona, 1997, (3 documents).
Muhugusa, M., et al., "ComScript : An Environment for the Implementation of Protocol Stacks and their Dynamic Reconfiguration", Dec. 1994.
Nelson, M., et al., The Data Compression Book, 2nd Edition, 1996, M&T Books, a division of MIS Press, Inc.
NetRanger User's Guide, 1996, WheelGroup Corporation.
NetScreen Command Line Reference Guide, 2000, P/N 093-0000-001 Rev A, NetScreen Technologies, Inc., USA.
NetScreen Command Line Reference Guide, 2000, P/N 093-0000-001 NetScreen Technologies, Inc., USA.
NetScreen Concepts and Examples ScreenOS Reference Guide, 1998-2001, Version 2.5 P/N 093-0039-000 Rev. A, NetScreen Technologies, Inc.
NetScreen Products Webpage, wysiwyg://body_bottom.3/http://www...een.com/products/products.html 1998-1999, NetScreen Technologies, Inc.
NetScreen WebUI, Reference Guide, Version 2.5.0 P/N 093-0040-000 Rev. A, 2000-2001, NetScreen Technologies, Inc.
NetStalker Installation and User's Guide, 1996, Version 1.0.2, Haystack Labs, Inc.
Niculescu, Dragos, "Survey of Active Network Research", Jul. 14, 1999, Rutgers University.
Nortel Northern Telecom, "ISDN Primary Rate User-Network Interface Specification", Aug. 1998.
Nygren, Erik, "The Design and Implementation of a High-Performance Active Network Node", Thesis, Feb. 1998, MIT.
Osbourne, E., "Morningstar Technologies SecureConnect Dynamic Firewall Filter User's Guide", Jun. 14, 1995, V. 1.4, Morning Star Technologies, Inc.
Padovano, Michael, "Networking Applications on UNIX System V Release 4," 1993 Prentice Hall, USA, (2 documents).
Pfeifer, T., "Automatic Conversion of Communication Media", 2000, GMD Research Series, Germany.
Pfeifer, T., "Automatic Conversion of Communication Media", Thesis, 1999, Technischen Universitat Berlin, Berlin.
Pfeifer, T., et al., "Applying Quality-of-Service Parametrization for Medium-to-Medium Conversion", Aug. 25-28, 1996, 8th IEEE Workshop on Local and Metropolitan Area Networks, Potsdam, Germany.
Pfeifer, T., "Micronet Machines—New Architectural Approaches for Multimedia End-Systems", 1993 Technical University of Berlin.
Pfeifer, T., "On the Convergence of Distributed Computing and Telecommunications in the Field of Personal Communications", 1995, KiVS, Berlin.
Pfeifer, T., "Speech Synthesis in the Intelligent Personal Communication Support System (IPCSS)", Nov. 2-3, 1995, 2nd 'Speak!' Workshop on Speech Generation in Multimodal Information Systems and Practical Applications.

(56) **References Cited**

OTHER PUBLICATIONS

Pfeifer, T., et al., "Generic Conversion of Communication Media for Supporting Personal Mobility", Nov. 25-27, 1996, Proc. of the Third COST 237 Workshop: Multimedia Telecommunications and Applications.
Pfeifer, T., et al., "Intelligent Handling of Communication Media", Oct. 29-31, 1997, 6th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS) Tunis.
Pfeifer, T., et al., "Resource Selection in Heterogeneous Communication Environments using the Teleservice Descriptor", Dec. 15-19, 1997, Proceedings from the 4th COST 237 Workshop: From Multimedia Services to Network Services, Lisboa.
Pfeifer, T., et al., Mobile Guide—Location-Aware Applications from the Lab to the Market, 1998, IDMS '98, LNCS 1483, pp. 15-28.
Pfeifer, T., et al., "The Active Store providing Quality Enhanced Unified Messaging", Oct. 20-22, 1998, 5th Conference on computer Communications, AFRICOM-CCDC '98, Tunis.
Pfeifer, T.,, et al., "*A Modular Location-Aware Service and Application Platform*", 1999, Technical University of Berlin.
Plagemann, T., et al., "*Evaluating Crucial Performance Issues of Protocol Configuration in DaCaPo*", 1994, University of Oslo.
Psounis, Konstantinos, "*Active Networks: Applications, Security, Safety, and Architectures*", First Quarter 1999, IEEE Communications Surveys.
Rabiner, Lawrence, "*Applications of Speech Recognition in the Area of Telecommunications*", 1997, IEEE.
Raman, Suchitra, et al, "*A Model, Analysis, and Protocol Framework for Soft State-based Communications*", Department of EECS, University of California, Berkeley.
Rogaway, Phillip, "*Bucket Hashing and its Application to Fast Message Authentication*", Oct. 13, 1997, Department of Computer Science, University of California.
Schneier, B., et al., "*Remote Auditing of Software Outputs Using a Trusted CoProcessor*", 1997, Elsevier Paper Reprint 1999.
Tennenhouse, D., et al., "*From Internet to ActiveNet*", Laboratory of Computer Science, MIT, 1996.
Tudor, P., "*Tutorial MPEG-2 Video Compression*", Dec. 1995, Electronics & Communication Engineering Journal.
US Copyright Webpage of Copyright Title, "*IPv6: the New Internet Protocol*", by Christian Huitema, 1998 Prentice Hall.
Van der Meer, et al., "*An Approach for a 4th Generation Messaging System*", Mar. 21-23, 1999, The Fourth International Symposium on Autonomous Decentralized Systems ISADS '99, Tokyo.
Van der Meer, Sven, "*Dynamic Configuration Management of the Equipment in Distributed Communication Environments*", Thesis, Oct. 6, 1996, Berlin, (3 documents).
Van Renesse, R. et al., "*Building Adaptive Systems Using Ensemble*", Cornell University Jul. 1997.
Venkatesan, R., et al., "*Threat-Adaptive Security Policy*", 1997, IEEE.
Wetherall, D., et al., "*The Active IP Option*", Sep. 1996, Proceedings of the 7th ACM SIGOPS European Workshop, Connemara, Ireland.
Welch, Terry, "*A Technique for High-Performance Data Compression*", 1984, Sperry Research Center, IEEE.
Zeletin, R. et al., "*Applying Location Aware Computing for Electronic Commerce: Mobile Guide*", Oct. 20-22, 1998, 5th Conference on Computer Communications, AFRICOM-CCDC '98, Tunis.
Zell, Markus, "*Selection of Converter Chains by Means of Quality of Service Analysis*", Thesis, Feb. 12, 1998, Technische Universitat Berlin.
Implicit Networks, Inc. v. *Advanced Micro Devices, Inc. et al.*; C08-0184 JLR; USDC for the Western District of Washington, Seattle Division.
Feb. 4, 2008 Plaintiff's Original Complaint.
Aug. 26, 2008 Defendant NVIDIA Corporation's Answer to Complaint.
Aug. 26, 2008 Defendant Sun Microsystems, Inc.'s Answer to Complaint.
Aug. 27, 2008 Defendant Advanced Micro Devices, Inc.'s Answer to Complaint for Patent Infringement.
Aug. 27, 2008 RealNetworks, Inc.'s Answer to Implicit Networks, Inc.'s Original Complaint for Patent Infringement, Affirmative Defenses, and Counterclaims.
Aug. 27, 2008 Intel Corp.'s Answer, Defenses and Counterclaims.
Aug. 27, 2008 Defendant RMI Corporation's Answer to Plaintiff's Original Complaint.
Sep. 15, 2008 Plaintiff's Reply to NVIDIA Corporation's Counterclaims.
Sep. 15, 2008 Plaintiff's Reply to Sun Microsystems Inc.'s Counterclaims.
Sep. 16, 2008 Plaintiff's Reply to RealNetworks, Inc.'s Counterclaims.
Sep. 16, 2008 Plaintiff's Reply to Intel Corp.'s Counterclaims.
Dec. 10, 2008 Order granting Stipulated Motion for Dismissal with Prejudice re NVIDIA Corporation, Inc.
Dec. 16, 2008 Defendants AMD, RealNetworks, RMI, and Sun's Motion to Stay Pending the Patent and Trademark Office's Reexamination of the '163 Patent.
Dec. 29, 2008 Order granting Stipulated Motion for Dismissal without Prejudice of Claims re Sun Microsystems, Inc.
Jan. 5, 2009 Plaintiff's Opposition to Defendants AMD, RealNetworks, RMI, and Sun's Motion to Stay Pending Reexamination and Exhibit A.
Jan. 9, 2009 Reply of Defendants AMD, RealNetworks, RMI, and Sun's Motion to Stay Pending the Patent and Trademark Office's Reexamination of the '163 Patent.
Feb. 9, 2009 Order Granting Stay Pending the United States Patent and Trademark Office's Reexamination of U.S. Pat. No. 6,629,163.
Feb. 17, 2009 Order Granting Stipulated Motion for Dismissal of Advanced Micro Devices, Inc. with Prejudice.
May 14, 2009 Order Granting Stipulated Motion for Dismissal of RMI Corporation with Prejudice.
Oct. 13, 2009 Order Granting Stipulated Motion for Dismissal of Claims Against and Counterclaims by Intel Corporation.
Oct. 30, 2009 Executed Order for Stipulated Motion for Dismissal of Claims Against and Counterclaims by RealNetworks, Inc.
Implicit Networks, Inc. v. *Microsoft Corp.*, C09-5628 HLR; USDC for the Northern District of California, San Francisco Division.
Nov. 30, 2009 Plaintiff's Original Complaint, *Implicit* v *Microsoft*, Case No. 09-5628.
Jan. 22, 2010 Order Dismissing Case, *Implicit* v *Microsoft*, Case No. 9-5628.
Implicit Networks, Inc. v. *Cisco Systems, Inc.*, C10-3606 HRL; USDC for the Northern District of California, San Francisco Division.
Aug. 16, 2010 Plaintiff's Original Complaint, *Implicit* v *Cisco*, Case No. 10-3606.
Nov. 22, 2010 Defendant Cisco Systems, Inc.'s Answer and Counterclaims, *Implicit* v *Cisco*, Case No. 10-3606.
Dec. 13, 2010 Plaintiff, Implicit Networks, Inc.'s, Answer to Counterclaims, *Implicit* v *Cisco*, Case No. 10-3606.
Oct. 4, 2011 Order of Dismissal with Prejudice, *Implicit* v *Cisco*, Case No. 10-3606.
Implicit Networks, Inc. v. *Citrix Systems, Inc.*, C10-3766 JL; USDC for the Northern District of California, San Francisco Division.
Aug. 24, 2010 Plaintiff's Original Complaint, *Implicit* v *Citrix*, Case No. 10-3766.
Dec. 1, 2010 Plaintiff's First Amended Complaint, *Implicit* v *Citrix*, Case No. 10-3766.
Jan. 14, 2011 Defendant Citrix Systems, Inc.'s Answer, Defense and Counter-complaint for Declaratory Judgment, *Implicit* v *Citrix*, Case No. 10-3766.
Feb. 18, 2011 Plaintiff, Implicit Networks, Inc.'s, Answer to Defendants Counterclaims, *Implicit* v *Citrix*, Case No. 10-3766.
May 2, 2011 Order of Dismissal, *Implicit* v *Citrix*, Case No. 10-3766.
Implicit Networks, Inc. v.*F5 Networks, Inc.*, C10-3365 JCS; USDC for the Northern District of California, San Francisco Division.
Jul. 30, 2010 Plaintiff's Original Complaint, *Implicit* v *F5*, Case No. 10-3365.

(56) **References Cited**

OTHER PUBLICATIONS

Oct. 13, 2010 Defendants' Answer and Counter-Complaint, *Implicit* v *F5*, Case No. 10-3365.
Nov. 3, 2010 Plaintiff's Answer to Counter-Complaint, *Implicit* v *F5*, Case No. 10-3365.
Dec. 10, 2010 Plaintiff's First Amended Complaint, *Implicit* v *F5*, Case No. 10-3365.
Jan. 14, 2011 Defendants' Answer to 1st Amended Complaint and Counterclaim, *Implicit* v *F5*, Case No. 10-3365.
Feb. 18, 2011 Plaintiff's Answer to F5's Amended Counter-Complaint, *Implicit* v *F5*, Case No. 10-3365.
Apr. 18, 2011 Defendants' Amended Answer to 1st Amended Complaint and Counter-Complaint, *Implicit* v *F5*, Case No. 10-3365.
May 5, 2011 Plaintiff's Answer to F5's Amended Counter-Complaint, *Implicit* v *F5*, Case No. 10-3365.
Jul. 22, 2011 F5 Networks, Inc.'s Invalidity Contentions, *Implicit* v *F5*, Case No. 10-3365.
Jul. 22, 2011 F5 Networks, Inc.'s Invalidity Contentions, Exhibit A, *Implicit* v *F5*, Case No. 10-3365, (31 documents).
Jul. 22, 2011 F5 Networks, Inc.'s Invalidity Contentions, Exhibit B, *Implicit* v *F5*, Case No. 10-3365.
Oct. 18, 2011 Joint Claim Construction & Pre-Hearing Statement (PR 4-3), *Implicit* v *F5*, Case No. 10-3365.
Oct. 18, 2011 Joint Claim Construction & Pre-Hearing Statement (PR 4-3) Exhibit A, *Implicit* v *F5*, Case No. 10-3365, (2 documents).
Nov. 28, 2011 Plaintiff's Opening Claim Construction Brief, *Implicit* v *F5*, Case No. 10-3365.
Nov. 29, 2011 Amended Joint Claim Construction & Pre-Hearing Statement, *Implicit* v *F5*, Case No. 10-3365.
Nov. 29, 2011 Amended Joint Claim Construction & Pre-Hearing Statement, Exhibit A, *Implicit* v *F5*, Case No. 10-3365.
Dec. 12, 2011 Defendants' Claim Construction Brief, *Implicit* v *F5*, Case No. 10-3365.
Dec. 19, 2011 Plaintiff's Reply to Defendants' (F5, HP, Juniper) Responsive Claim Construction Brief (4-5), *Implicit* v *F5*, Case No. 10-3365.
Jan. 27, 2012 Transcript of Proceeding Held on Jan. 17, 2012; *Implicit* v *F5*, Case No. 10-3365.
Jan. 27, 2012 Transcript of Proceeding Held on Jan. 18, 2012; *Implicit* v *F5*, Case No. 10-3365.
Jan. 27, 2012 Transcript of Proceeding Held on Jan. 19, 2012; *Implicit* v *F5*, Case No. 10-3365.
Feb. 29, 2012 Claim Construction Order.
Aug. 15, 2012 Storer Invalidity Report.
Sep. 10, 2012 Implicit's Expert Report of Scott M. Nettles.
Mar. 13, 2013 Order Granting Defendants' Motion for Summary Judgment.
Apr. 9, 2013 Notice of Appeal to the Federal Circuit.
Implicit Networks, Inc. v. *Hewlett-Packard Company*, C10-3746 JCS: USDC for the Northern District of California, San Francisco Division.
Aug. 23, 2010 Plaintiff's Original Complaint, *Implicit* v *HP*, Case No. 10-3746.
Nov. 23, 2010 Plaintiff's First Amended Complaint, *Implicit* v *HP*, Case No. 10-3746.
Nov. 14, 2011 Defendant HP's Answer and Counterclaims, *Implicit* v *HP*, Case No. 10-3746.
Feb. 18, 2011 Implicit Networks, Inc.'s Answer to HP Counterclaims, *Implicit* v *HP*, Case No. 10-3746.
May 10, 2011 Plaintiff's Amended Disclosure of Asserted Claims and Infringement Contentions, Case No. 10-3746.
Jun. 30, 2011 Defendant HP Company's Invalidity Contentions, *Implicit* v *HP*, Case No. 10-3746.
Jun. 30, 2011 Defendant HP Company's Invalidity Contentions, A1-14, *Implicit* v *HP*, Case No. 10-3746.
Jun. 30, 2011 Defendant HP Company's Invalidity Contentions, B1-21, *Implicit* v *HP*, Case No. 10-3746.
Implicit Networks, Inc. v. *Juniper Networks*, C10-4234 EDL: USDC for the Northern District of California, San Francisco Division.
Sep. 20, 2010 Plaintiff's Original Complaint, *Implicit* v *Juniper*, Case No. 10-4234.
Nov. 12, 2010 Juniper Network's Motion to Dismiss for Failure to State a Claim Under Rule 12(B)(6): Memorandum of Points and Authorities; *Implicit* v *Juniper*, Case No. 10-4234.
Nov. 12, 2010 Juniper Network's Request for Judicial Notice in Support of its Motion to Dismiss for Failure to State a Claim Under Rule 12(B)(6): Memorandum of Points and Authorities; *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 1, 2010 First Amended Complaint; *Implicit* v *Juniper*, Case No. 10-4234.
Jan. 18, 2011 Juniper Networks, Inc.'s Answer and Affirmative Defenses to 1st Amended Complaint, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 18, 2011 Plaintiff's Answer to Defendant's Counterclaims, *Implicit* v *Juniper*, Case No. 10-4234.
May 23, 2011 Plaintiff's Disclosure of Asserted Claims and Infringement Contentions, *Implicit* v *Juniper*, Case No. 10-4234.
Nov. 15, 2011 Plaintiff's Amended Disclosure of Asserted Claim and Infringement Contentions, *Implicit* v *Juniper*, Case No. 10-4234.
Nov. 28, 2011 Spencer Hosie Declaration in Support of Plaintiff's Opening Claim Construction Brief), *Implicit* v *Juniper*, Case No. 10-4234.
Nov. 28, 2011 Spencer Hosie Declaration in Support of Plaintiff's Opening Claim Construction Brief Exhibit E, *Implicit* v *Juniper*, Case No. 10-4234.
Nov. 28, 2011 Spencer Hosie Declaration in Support of Plaintiff's Opening Claim Construction Brief Exhibit J, *Implicit* v *Juniper*, Case No. 10-4234.
Nov. 28, 2011 Spencer Hosie Declaration in Support of Plaintiff's Opening Claim Construction Brief Exhibit K, *Implicit* v *Juniper*, Case No. 10-4234.
Nov. 28, 2011 Spencer Hosie Declaration in Support of Plaintiff's Opening Claim Construction Brief Exhibits M-O, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit B, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit F, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit N, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit P, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit Q, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit S., *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit T-1, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit T-2, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit T-3, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit T-4, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit U, *Implicit* v *Juniper*, Case No. 10-4234.

(56) **References Cited**

OTHER PUBLICATIONS

Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit V, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit W, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit X, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit Y-1, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit Y-2, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit Y-3, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit Y-4, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit Z, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Spencer Hosie Declaration in Support of Plaintiff's Reply Claim Construction Brief, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Spencer Hosie Declaration in Support of Plaintiff's Reply Claim Construction Brief, Exhibit P, *Implicit* v *Juniper*, Case No. 10-4234.
Jan. 10, 2012 Plaintiff's Jan. 10, 2012 Amended Disclosure of Asserted Claims and Infringement Contentions, *Implicit* v *Juniper*, Case No. 10-4234.
Feb. 10, 2012 Juniper Networks, Inc.'s Supplemental Invalidity Contentions, *Implicit* v *Juniper*, Case No. 10-4234.
Feb. 10, 2012 Juniper Networks, Inc.'s Supplemental Invalidity Contentions, Exhibit A1, *Implicit* v *Juniper*, Case No. 10-4234.
Feb. 10, 2012 Juniper Networks, Inc.'s Supplemental Invalidity Contentions, Exhibit A2, *Implicit* v *Juniper*, Case No. 10-4234.
Feb. 10, 2012 Juniper Networks, Inc.'s Supplemental Invalidity Contentions, Exhibit A3, *Implicit* v *Juniper*, Case No. 10-4234.
Feb. 10, 2012 Juniper Networks, Inc.'s Supplemental Invalidity Contentions, Exhibit A4, *Implicit* v *Juniper*, Case No. 10-4234.
Feb. 10, 2012 Juniper Networks, Inc.'s Supplemental Invalidity Contentions, Exhibit B1, *Implicit* v *Juniper*, Case No. 10-4234.
Feb. 29, 2012 Plaintiff's Feb. 29, 2012 Amended Disclosure of Asserted Claims and Infringement Contentions, *Implicit* v *Juniper*, Case No. 10-4234.
Apr. 6, 2012 Plaintiff's Apr. 6, 2012 Amended Disclosure of Asserted Claims and Infringement Contentions, *Implicit* v *Juniper*, Case No. 10-4234.
Apr. 9, 2012 Plaintiff's Apr. 9, 2012 Amended Disclosure of Asserted Claims and Infringement Contentions, *Implicit* v *Juniper*, Case No. 10-4234.
Sep. 11, 2012 Implicit's Expert Report of Scott Nettles.
Nov. 9, 2012 Juniper's Notice of Motion and Memorandum of Law ISO Motion for Summary Judgment or, in the alternative, for Partial Summary Judgment, on the Issue of Invalidity.
Nov. 9, 2012 Exhibit 2 to Declaration in support of Juniper's Motion for Summary Judgment—Calvert Expert Report.
Nov. 9, 2012 Exhibit 3 to Declaration in support of Juniper's Motion for Summary Judgment—Calvert Supplemental Expert Report.
Nov. 26, 2012 Implicit Opposition to Juniper's and F5 Motion on Invalidity.
Nov. 26, 2012 Exhibit A to Hosie Declaration—Aug. 27, 2012 Excerpts from David Blaine deposition.
Nov. 26, 2012 Exhibit B to Hosie Declaration—Oct. 25, 2012 Excerpts from Kenneth Calvert Deposition.
Nov. 26, 2012 Exhibit C to Hosie Declaration—Aug. 15, 2012 Excerpts from Kenneth Calvert Expert Report.
Nov. 26, 2012 Exhibit D to Hosie Declaration—U.S. Pat. No. 6,651,099 to Dietz et al.
Nov. 26, 2012 Exhibit E to Hosie Declaration—Understanding Packet-Based and Flow-Based Forwarding.
Nov. 26, 2012 Exhibit F to Hosie Declaration—Wikipedia on Soft State.
Nov. 26, 2012 Exhibit G to Hosie Declaration—Sprint Notes.
Nov. 26, 2012 Exhibit H to Hosie Declaration—Implicit's Supplemental Response to Juniper's 2nd Set of Interrogatories.
Nov. 26, 2012 Exhibit I to Hosie Declaration—U.S. Pat. No. 7,650,634 (Zuk).
Other Implicit Networks, Inc. Prosecution Matters.
U.S. Appl. No. 11/933,022 Utility Application filed Oct. 31, 2007.
U.S. Appl. No. 11/933,022 Preliminary Amendment filed Feb. 19, 2008.
U.S. Appl. No. 11/933,022 Office Action dated Jun. 24, 2009.
U.S. Appl. No. 11/933,022 Amendment filed Sep. 24, 2009.
U.S. Appl. No. 11/933,022 Office Action dated Dec. 11, 2009.
U.S. Appl. No. 11/933,022 Amendment and Response dated Jan. 29, 2010.
U.S. Appl. No. 11/933,022 Notice of Allowance dated Mar. 2, 2010.
U.S. Appl. No. 11/933,022 Issue Notification dated May 4, 2010.
U.S. Appl. No. 10/636,314 Utility Application filed Aug. 6, 2003.
U.S. Appl. No. 10/636,314 Office Action dated Apr. 7, 2008.
U.S. Appl. No. 10/636,314 Response to Restriction Requirement dated Aug. 5, 2008.
U.S. Appl. No. 10/636,314 Office Action dated Oct. 3, 2008.
U.S. Appl. No. 10/636,314 Response to Office Action dated Apr. 3, 2009.
U.S. Appl. No. 10/636,314 Notice of Non-Compliant Amendment dated May 4, 2009.
U.S. Appl. No. 10/636,314 Amendment to Office Action Response dated Jun. 4, 2009.
U.S. Appl. No. 10/636,314 Notice of Non-Compliant Amendment dated Jun. 12, 2009.
U.S. Appl. No. 10/636,314 Amendment to Office Action dated Jul. 10, 2009.
U.S. Appl. No. 10/636,314 Final Rejection Office Action dated Oct. 21, 2009.
U.S. Appl. No. 10/636,314 Amendment after Final Office Action dated Dec. 14, 2009.
U.S. Appl. No. 10/636,314 Advisory Action dated Jan. 11, 2010.
U.S. Appl. No. 10/636,314 Notice of Non-Compliant Amendment dated Jan. 11, 2010.
U.S. Appl. No. 10/636,314 Supplemental Amendment and Response dated Mar. 13, 2010.
U.S. Appl. No. 10/636,314 Office Action dated May 11, 2010.
U.S. Appl. No. 10/636,314 Amendment and Response dated Sep. 13, 2010.
U.S. Appl. No. 10/636,314 Final Rejection dated Nov. 24, 2010.
U.S. Appl. No. 10/636,314 Notice of Appeal dated May 19, 2011.
U.S. Appl. No. 10/636,314 Amendment and Request for Continued Examination dated Jul. 19, 2011.
U.S. Appl. No. 10/636,314 Notice of Allowance dated Sep. 13, 2011.
U.S. Appl. No. 10/636,314 Notice of Allowance dated Sep. 19, 2011.
U.S. Appl. No. 10/636,314 Issue Notification dated Oct. 19, 2011.
U.S. Appl. No. 09/474,664 Utility Application filed Dec. 29, 1999.
U.S. Appl. No. 09/474,664 Office Action dated Sep. 23, 2002.
U.S. Appl. No. 09/474,664 Amendment and Response dated Feb. 24, 2003.
U.S. Appl. No. 09/474,664 Notice of Allowance dated May 20, 2003.
U.S. Appl. No. 90/010,356 Request for Ex Parte Reexamination dated Dec. 15, 2008.
U.S. Appl. No. 90/010,356 Office Action Granting Reexamination dated Jan. 17, 2009.
U.S. Appl. No. 90/010,356 First Office Action dated Jul. 7, 2009.
U.S. Appl. No. 90/010,356 First Office Action Response dated Sep. 1, 2009.

(56) **References Cited**

OTHER PUBLICATIONS

U.S. Appl. No. 90/010,356 Patent Owner Interview Summary dated Oct. 23, 2009.
U.S. Appl. No. 90/010,356 Office Action Final dated Dec. 4, 2009.
U.S. Appl. No. 90/010,356 Amendment and Response to Office Action dated Dec. 18, 2009.
U.S. Appl. No. 90/010,356 Amendment and Response to Office Action dated Jan. 4, 2010.
U.S. Appl. No. 90/010,356 Advisory Action dated Jan. 21, 2010.
U.S. Appl. No. 90/010,356 Amendment and Response to Advisory Action dated Feb. 8, 2010.
U.S. Appl. No. 90/010,356 Notice of Intent to Issue a Reexam Certificate dated Mar. 2, 2010.
U.S. Appl. No. 90/010,356 Reexamination Certificate Issued dated Jun. 22, 2010.
U.S. Appl. No. 95/000,659 Inter Partes Reexam Request dated Feb. 13, 2012.
U.S. Appl. No. 95/000,659 Order Granting Reexamination dated Apr. 3, 2012.
U.S. Appl. No. 95/000,659 Office Action dated Apr. 3, 2012.
U.S. Appl. No. 95/000,659 Office Action Response dated Jun. 4, 2012 (including Exhibits 1 & 2), (4 documents).
U.S. Appl. No. 95/000,659 Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012.
U.S. Appl. No. 95/000,659 Appendix R-1 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Declaration of Prof. Dr. Bernhard Plattner).
U.S. Appl. No. 95/000,659 Appendix R-2 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Prof. Dr. Bernhard Plattner CV).
U.S. Appl. No. 95/000,659 Appendix R-3 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Listing of Publications to Prof. Dr. Bernhard Plattner updated Feb. 2012).
U.S. Appl. No. 95/000,659 Appendix R-4 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012(Office Action Granting Reexamination in 95/000,660 dated May 10, 2012).
U.S. Appl. No. 95/000,659 Appendix R-5 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Office Action in 95/000,660 dated May 10, 2012).
U.S. Appl. No. 95/000,659 Appendix R-6 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Implicit Networks, Inc. U.S. Pat. No. 6,629,163 Claims Chart).
U.S. Appl. No. 95/000,659 Appendix R-7 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Internet Protocol DARPA Internet Program Protocol Specification dated Sep. 1991).
U.S. Appl. No. 95/000,659 Appendix R-8 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Atkinson, IP Encapsulating Security Payload (ESP) dated Aug. 1995).
U.S. Appl. No. 95/000,659 Appendix R-9 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Claim Construction Order dated Feb. 29, 2012).
U.S. Appl. No. 95/000,659 Appendix R-10-1 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (vol. I of Edward Balassanian Deposition Transcript dated May 30, 2012).
U.S. Appl. No. 95/000,659 Appendix R-10-2 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (vol. II of Edward Balassanian Deposition Transcript dated May 31, 2012).
U.S. Appl. No. 95/000,659 Appendix R-10-3 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (vol. III of Edward Balassanian Deposition Transcript dated Jun. 7, 2012).
U.S. Appl. No. 95/000,659 Appendix R-10-4 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (vol. IV of Edward Balassanian Deposition Transcript dated Jun. 8, 2012).
U.S. Appl. No. 95/000,659 Appendix R-11 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Implicit Networks, Inc.'s Response to Juniper Networks, Inc.'s First Set of Requests for Admission 1-32).
U.S. Appl. No. 95/000,659 Action Closing Prosecution dated Oct. 1, 2012.
U.S. Appl. No. 95/000,659 Petition to Withdraw and Reissue Action Closing Prosecution dated Nov. 20, 2012.
U.S. Appl. No. 95/000,659 Patent Owner Comments to Action Closing Prosecution dated Dec. 3, 2012.
U.S. Appl. No. 95/000,659 Opposition to Petition dated Dec. 17, 2012.
U.S. Appl. No. 95/000,659 Third Party Comments to Action Closing Prosecution dated Jan. 2, 2013.
U.S. Appl. No. 95/000,660 Inter Partes Reexam Request dated Mar. 2, 2012.
U.S. Appl. No. 95/000,660 Order Granting Reexamination dated May 10, 2012.
U.S. Appl. No. 95/000,660 Office Action dated May 10, 2012.
U.S. Appl. No. 95/000,660 Response to Office Action dated Jul. 10, 2012 (including Exhibits 1 and 2).
U.S. Appl. No. 95/000,660 Third Party Comments to Office After Patent Owner's Response dated Aug. 8, 2012 (including Revised Comments).
U.S. Appl. No. 95/000,660 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Declaration of Prof. Dr. Bernhard Plattner).
U.S. Appl. No. 95/000,660 Appendix R-1 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Prof. Dr. Bernhard Plattner CV).
U.S. Appl. No. 95/000,660 Appendix R-3 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Listing of Publications to Prof. Dr. Bernhard Plattner updated Feb. 2012).
U.S. Appl. No. 95/000,660 Appendix R-4 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012(Office Action Granting Reexamination in 95/000,660 dated May 10, 2012).
U.S. Appl. No. 95/000,660 Appendix R-5 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Office Action in 95/000,660 dated May 10, 2012).
U.S. Appl. No. 95/000,660 Appendix R-6 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Implicit Networks, Inc. U.S. Pat. No. 6,629,163 Claims Chart).
U.S. Appl. No. 95/000,660 Appendix R-7 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Internet Protocol DARPA Internet Program Protocol Specification dated Sep. 1991).
U.S. Appl. No. 95/000,660 Appendix R-8 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Atkinson, IP Encapsulating Security Payload (ESP) dated Aug. 1995).
U.S. Appl. No. 95/000,660 Appendix R-9 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Claim Construction Order dated Feb. 29, 2012).
U.S. Appl. No. 95/000,660 Appendix R-10 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (vol. I-IV of Edward Balassanian Deposition Transcript dated May 30, 2012).
U.S. Appl. No. 95/000,660 Appendix R-11 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Shacham, A., et al, "*IP Payload Compression Protocol*", Network Working Group, RFC 3173 Sep. 2001).
U.S. Appl. No. 95/000,660 Appendix R-12 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Shacham, A., et al, "*IP Payload Compression Protocol*", Network Working Group, RFC 2393 Dec. 1998).

(56) **References Cited**

OTHER PUBLICATIONS

U.S. Appl. No. 95/000,660 Appendix R-13 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 ('163 Pfeiffer Claim Chart).
U.S. Appl. No. 95/000,660 Appendix R-14 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Ylonen, T., "*SSH Transport Layer Protocol*", Network Working Group—Draft Feb. 22, 1999).
U.S. Appl. No. 95/000,660 Appendix R-15 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Dommety, G., "*Key and Sequence Number Extensions to GRE*", Network Working Group, RFC 2890 Sep. 2000).
U.S. Appl. No. 95/000,660 Appendix R-16 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Monsour, R., et al, "*Compression in IP Security*" Mar. 1997).
U.S. Appl. No. 95/000,660 Appendix R-17 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Friend, R., Internet Working Group RFC 3943 dated Nov. 2004 *Transport Layer Security Protocol Compression Using Lempel-Ziv-Stac*).
U.S. Appl. No. 95/000,660 Appendix R-18 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012(Implicit Networks, Inc.'s Response to Juniper Networks, Inc.'s First Set of Requests for Admission 1-32).
U.S. Appl. No. 95/000,660 Revised—Third Party Comments to Office After Patent Owner's Response dated Nov. 2, 2012.
U.S. Appl. No. 95/000,660 Action Closing Prosecution dated Dec. 21, 2012.
U.S. Appl. No. 95/000,660 Comments to Action Closing Prosecution dated Feb. 21, 2013 (including Dec of Dr. Ng).
U.S. Appl. No. 95/000,660 Third Party Comments to Action Closing Prosecution dated Mar. 25, 2013.
PCT/US00/33634—PCT application (WO 01/2077 A2—Jul. 12, 2001).
PCT/US00/33634—Written Opinion (WO 01/50277 A3—Feb. 14, 2002).
PCT/US00/33634—International Search Report (dated Oct. 9, 2001).
PCT/US00/33634—Response to Official Communication dated Dec. 7, 2001 (dated Mar. 21, 2002).
PCT/US00/33634—International Preliminary Examination Report (dated Apr. 8, 2002).
PCT/US00/33634—Official Communication (dated Jan. 24, 2003).
PCT/US00/33634—Response to Official Communication dated Jan. 24, 2003 (dated Mar. 12, 2003).
PCT/US00/33634—Official Communication (dated May 13, 2004).
PCT/US00/33634—Response to Summons to Attend Oral Proceeding dated May 13, 2004 (dated Oct. 9, 2004).
PCT/US00/33634—Decision to Refuse a European Patent application (dated Nov. 12, 2004).
PCT/US00/33634—Minutes of the oral proceedings before the Examining Division (dated Oct. 12, 2004).
PCT/US00/33634—Closure of the procedure in respect to Application No. 00984234.5-2212 (dated Feb. 22, 2005).
May 3, 2013 Expert Report of Dr. Alfonso Cardenas Regarding Validity of U.S. Pat. Nos. 6,877,006; 7,167,864; 7,720,861; and 8,082,268, (6 documents).
Expert Report of Dr. Alfonso Cardenas Regarding Validity of U.S. Pat. No. 7,167,864, (3 documents).
"InfoReports User Guide: Version 3.3.1;" Platinum Technology, Publication No. PRO-X-331-UG00-00, printed Apr. 1998; pp. 1-430.

* cited by examiner

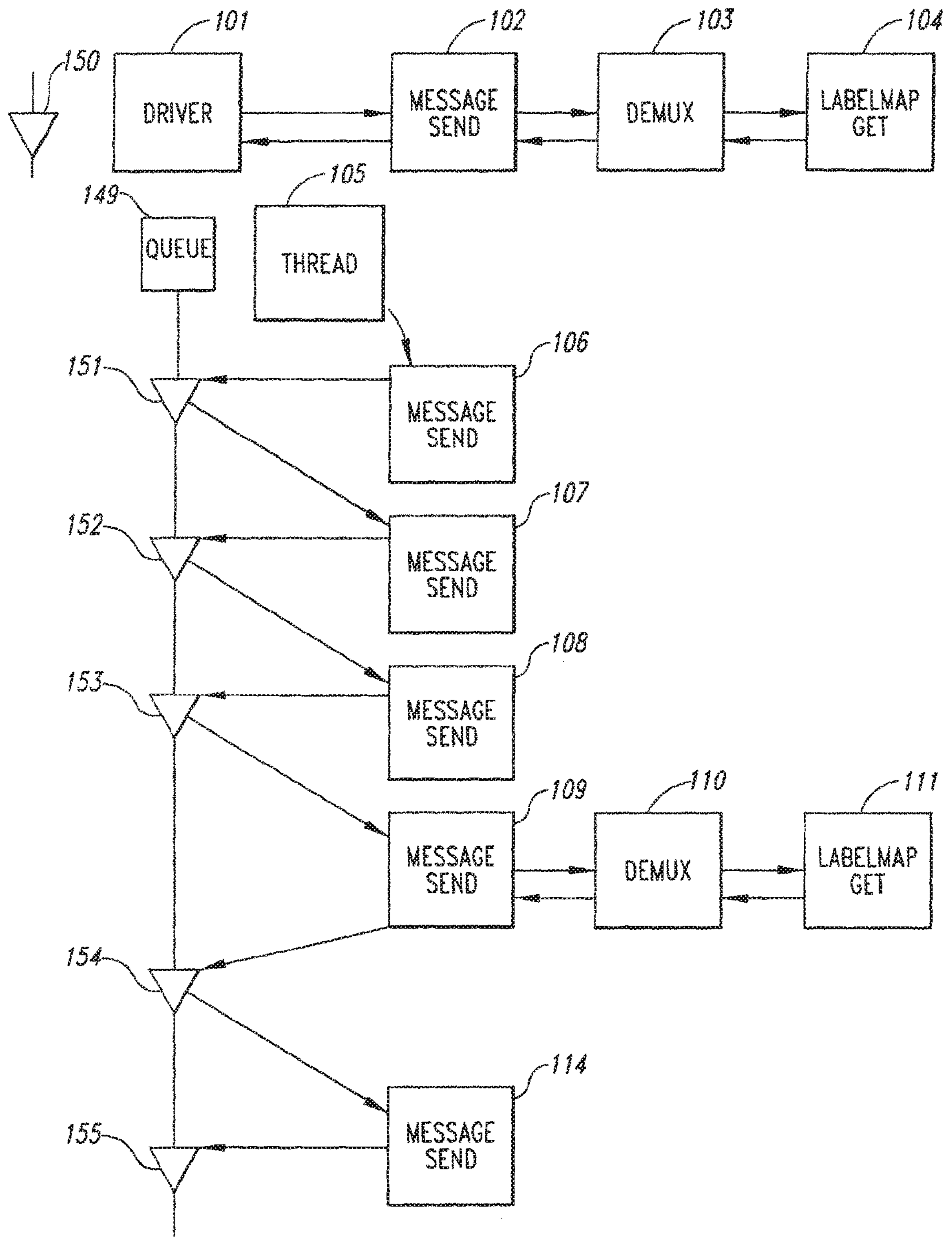
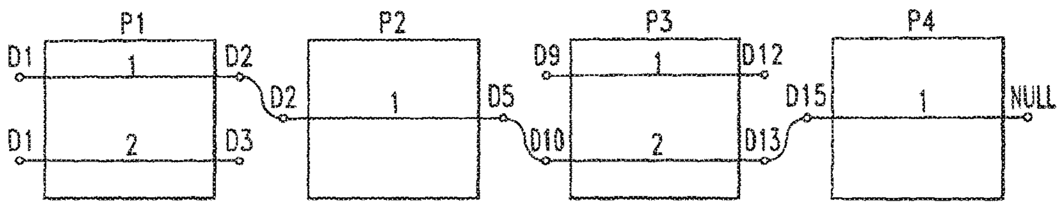
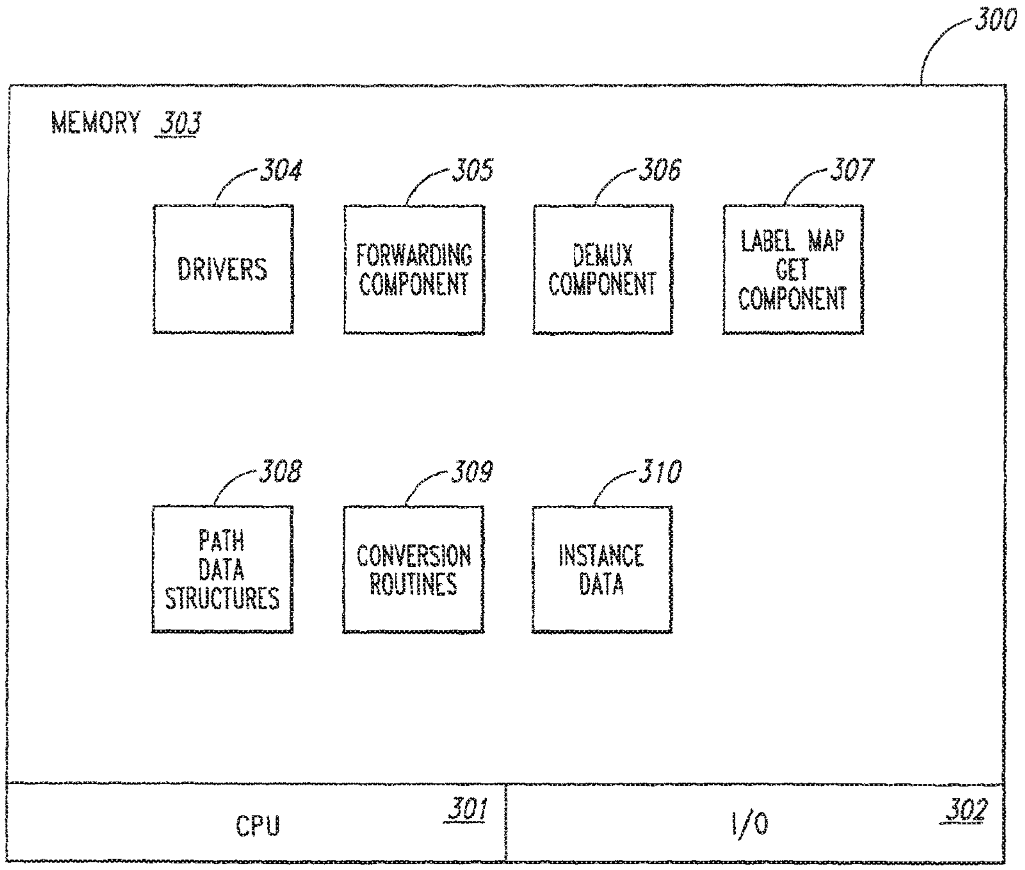

101
DRIVER
102
MESSAGE SEND
103
DEMUX
104
LABELMAP GET
150
149
QUEUE
105
THREAD
106
MESSAGE SEND
151
107
MESSAGE SEND
152
108
MESSAGE SEND
153
109
MESSAGE SEND
110
DEMUX
111
LABELMAP GET
154
114
MESSAGE SEND
155

Fig. 1



P1
P2
P3
P4
D1
1
D2
D1
2
D3
D2
1
D5
D9
1
D12
D10
2
D13
D15
1
NULL

Fig. 2



300
MEMORY 303
304
DRIVERS
305
FORWARDING COMPONENT
306
DEMUX COMPONENT
307
LABEL MAP GET COMPONENT
308
PATH DATA STRUCTURES
309
CONVERSION ROUTINES
310
INSTANCE DATA
CPU 301
I/O 302

Fig. 3

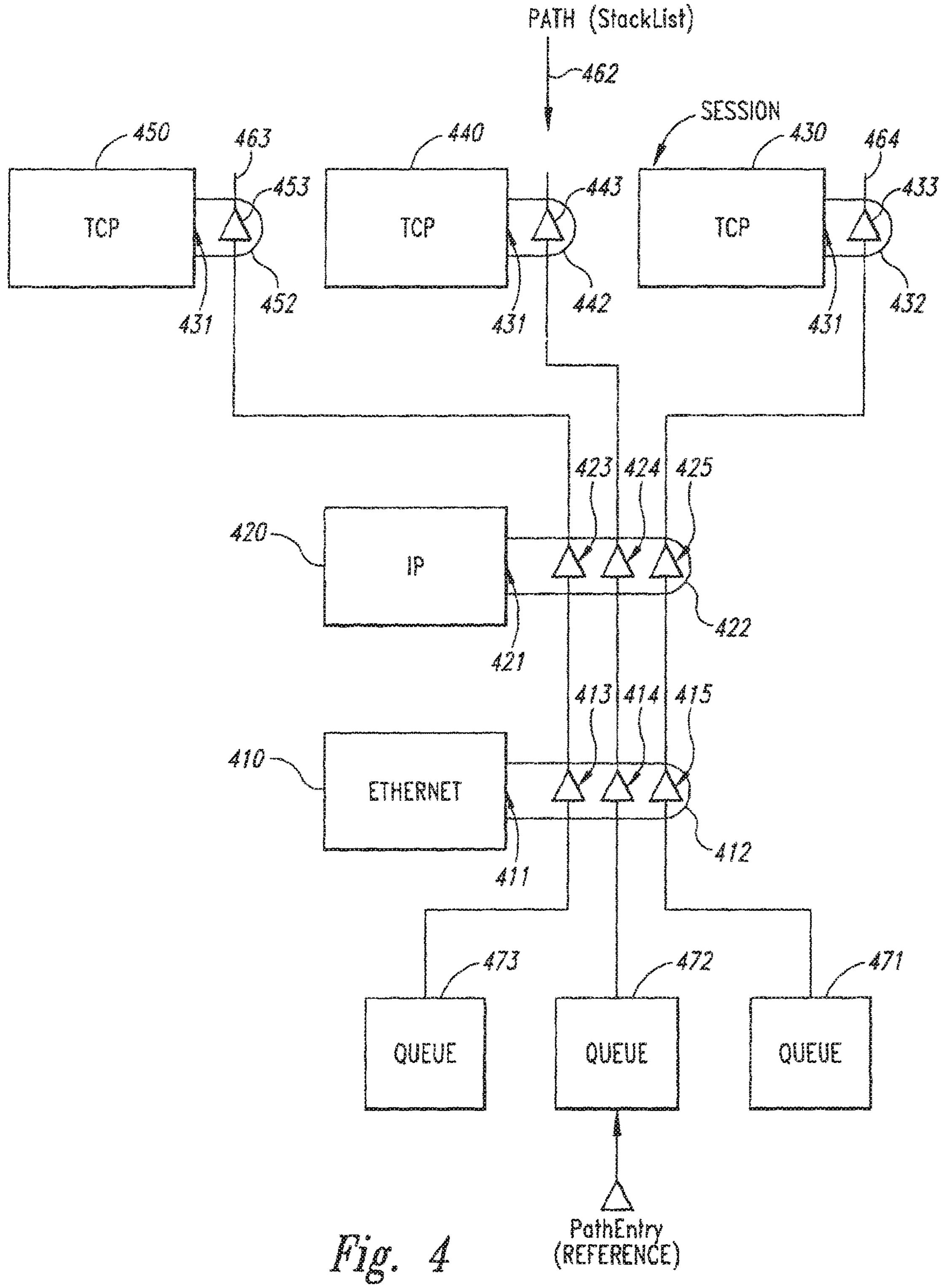

PATH (StackList)
462
SESSION
450
463
453
TCP
431
452
440
443
TCP
431
442
430
464
433
TCP
431
432
423
424
425
420
IP
421
422
413
414
415
410
ETHERNET
411
412
473
472
471
QUEUE
QUEUE
QUEUE
PathEntry (REFERENCE)

Fig. 4

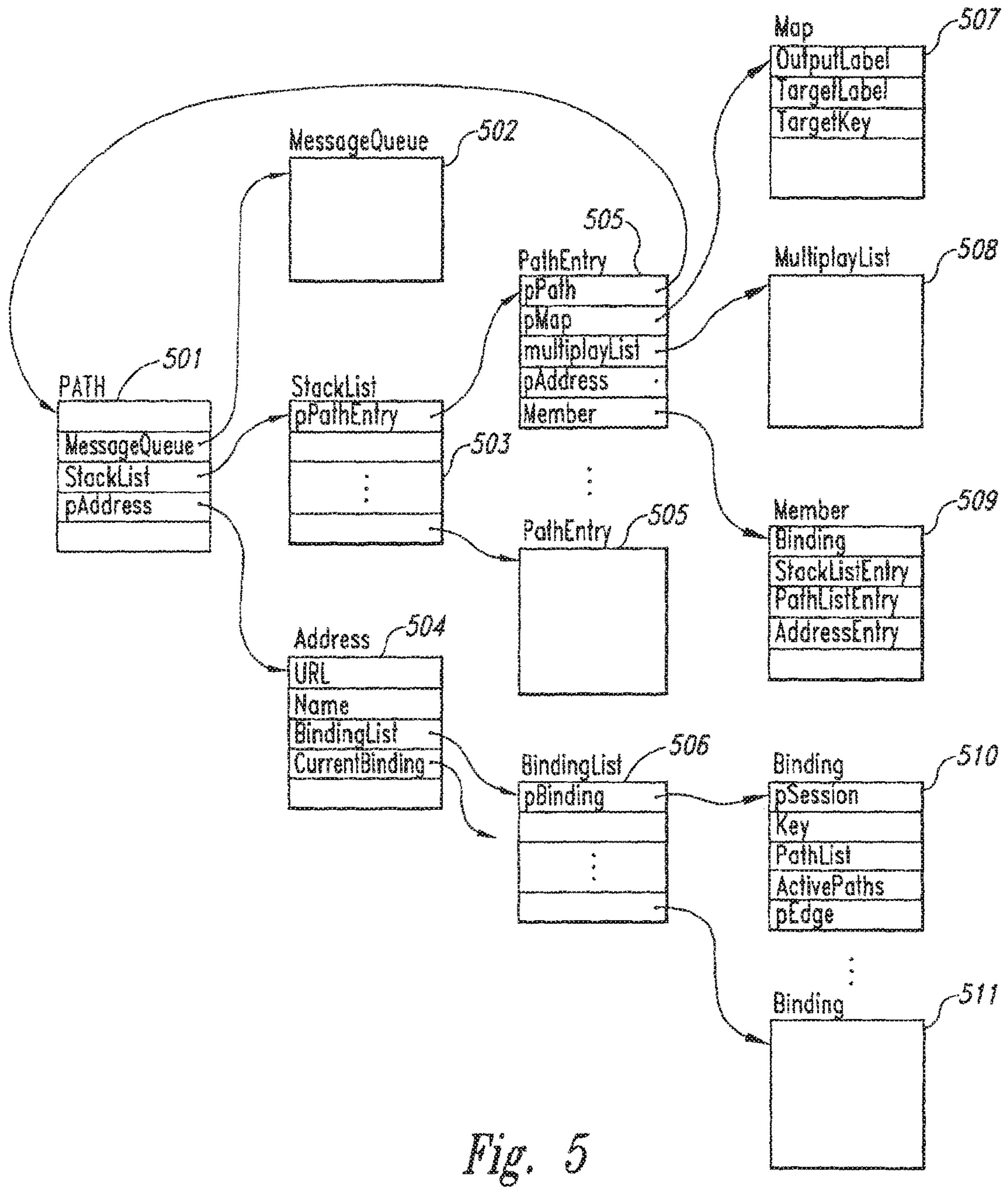
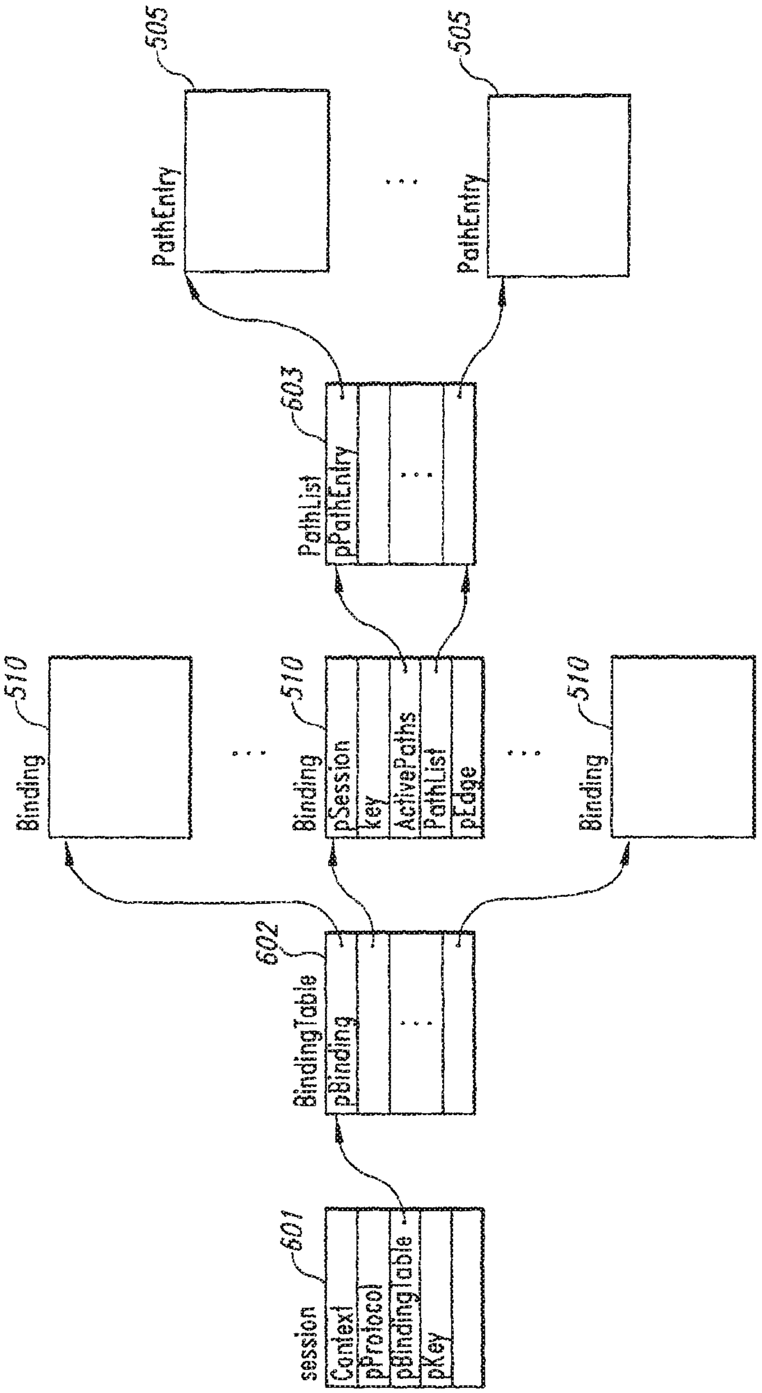

Map
507
OutputLabel
TargetLabel
TargetKey
MessageQueue
502
505
PathEntry
pPath
pMap
multiplayList
pAddress
Member
MultiplayList
508
PATH
501
MessageQueue
StackList
pAddress
StackList
pPathEntry
503
PathEntry
505
Member
509
Binding
StackListEntry
PathListEntry
AddressEntry
Address
504
URL
Name
BindingList
CurrentBinding
BindingList
506
pBinding
Binding
510
pSession
Key
PathList
ActivePaths
pEdge
Binding
511

Fig. 5



session
601
Context
pProtocol
pBindingTable
pKey
BindingTable
602
pBinding
Binding
510
pSession
key
ActivePaths
PathList
pEdge
PathList
603
pPathEntry
PathEntry
505

Fig. 6

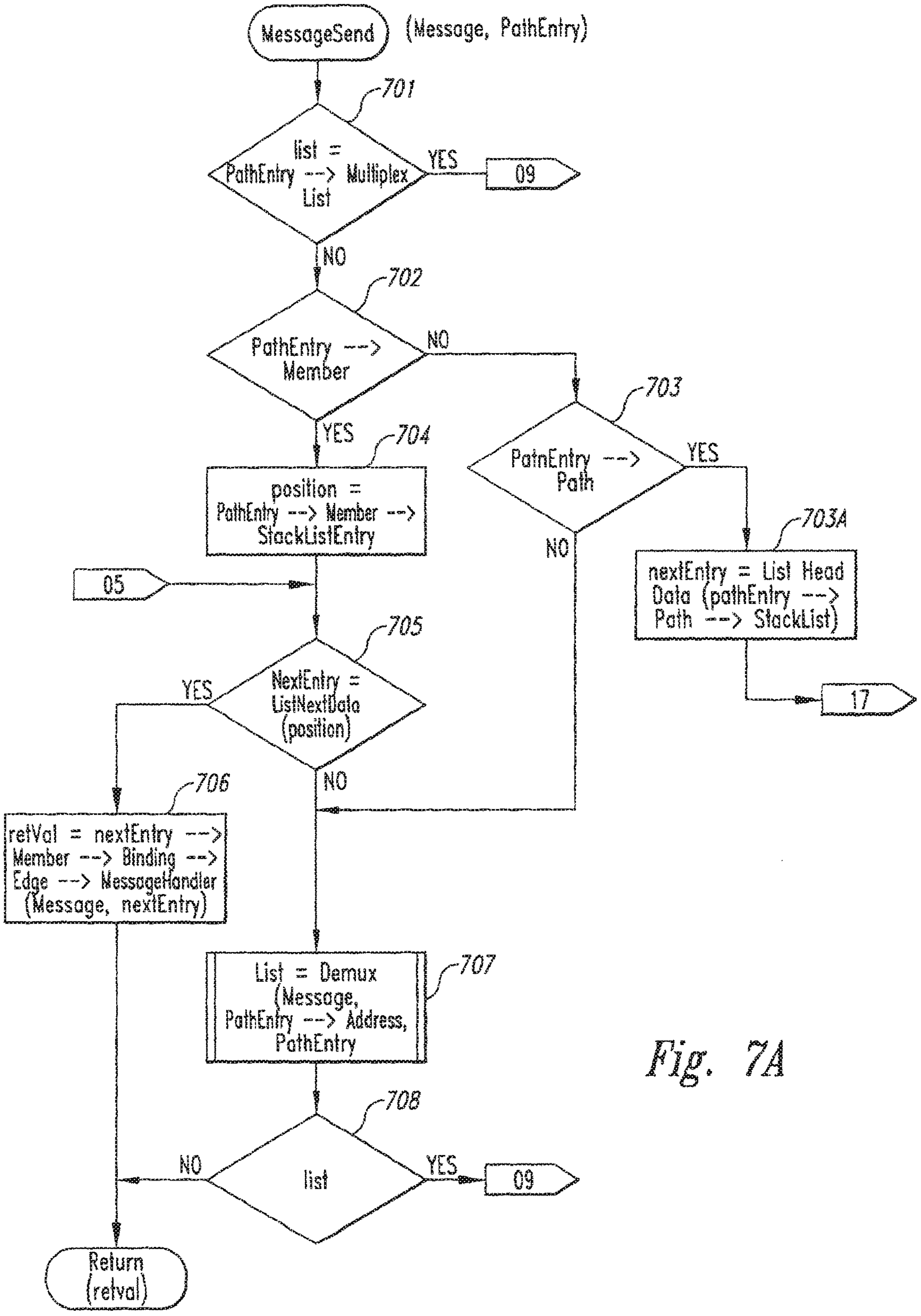

MessageSend
(Message, PathEntry)
701
list = PathEntry --> Multiplex List
YES
09
NO
702
PathEntry --> Member
NO
703
PatnEntry --> Path
YES
703A
nextEntry = List Head Data (pathEntry --> Path --> StackList)
17
NO
YES
704
position = PathEntry --> Member --> StackListEntry
05
705
NextEntry = ListNextData (position)
YES
NO
706
retVal = nextEntry --> Member --> Binding --> Edge --> MessageHandler (Message, nextEntry)
707
List = Demux (Message, PathEntry --> Address, PathEntry
708
list
NO
YES
09
Return (retval)

Fig. 7A

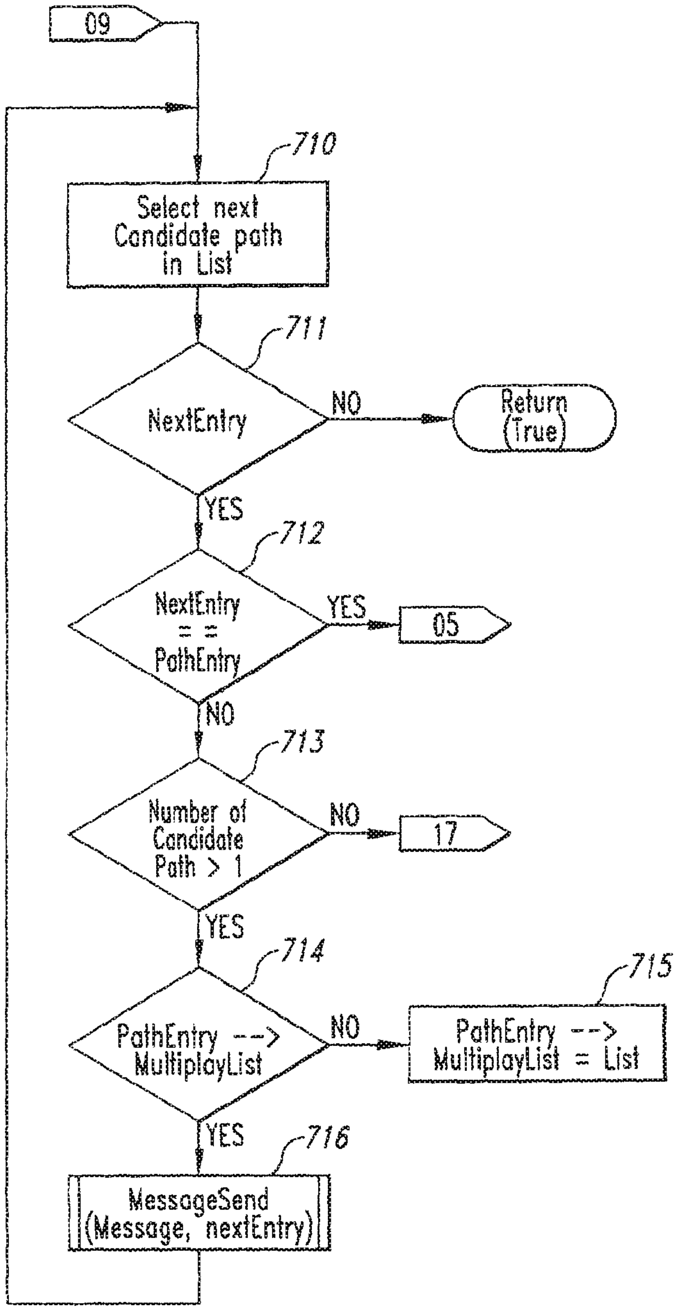

09
710
Select next Candidate path in List
711
NextEntry
NO
Return (True)
YES
712
NextEntry == PathEntry
YES
05
NO
713
Number of Candidate Path > 1
NO
17
YES
714
PathEntry --> MultiplayList
NO
715
PathEntry --> MultiplayList = List
YES
716
MessageSend (Message, nextEntry)

Fig. 7B

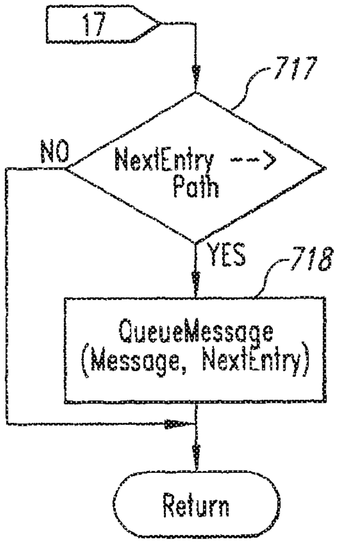

17
717
NextEntry --> Path
NO
YES
718
QueueMessage (Message, NextEntry)
Return

Fig. 7C

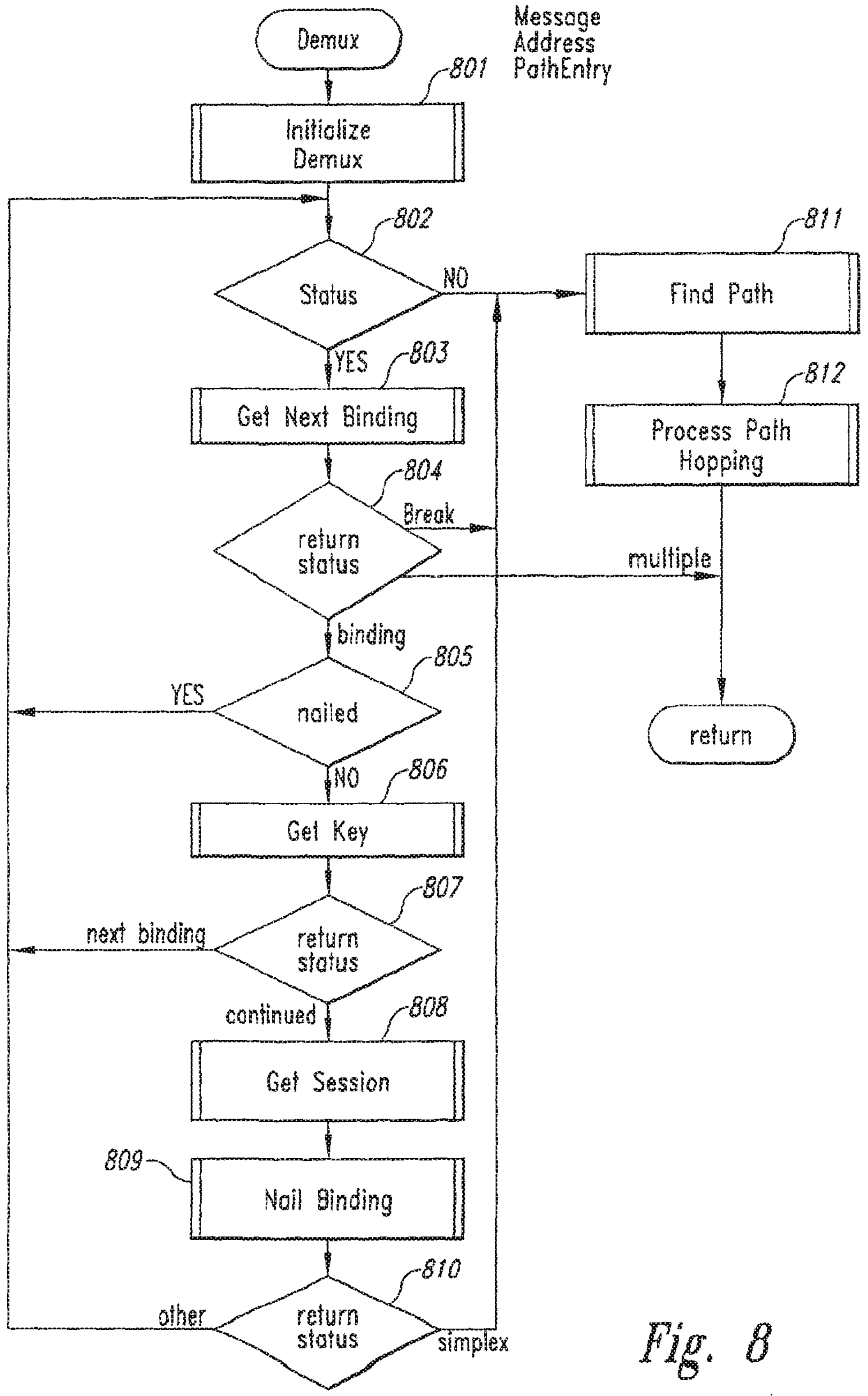

Demux
Message
Address
PathEntry
801
Initialize Demux
802
Status
NO
YES
811
Find Path
803
Get Next Binding
812
Process Path Hopping
804
Break
return status
multiple
binding
805
YES
nailed
return
NO
806
Get Key
807
next binding
return status
continued
808
Get Session
809
Nail Binding
810
other
return status
simplex

Fig. 8

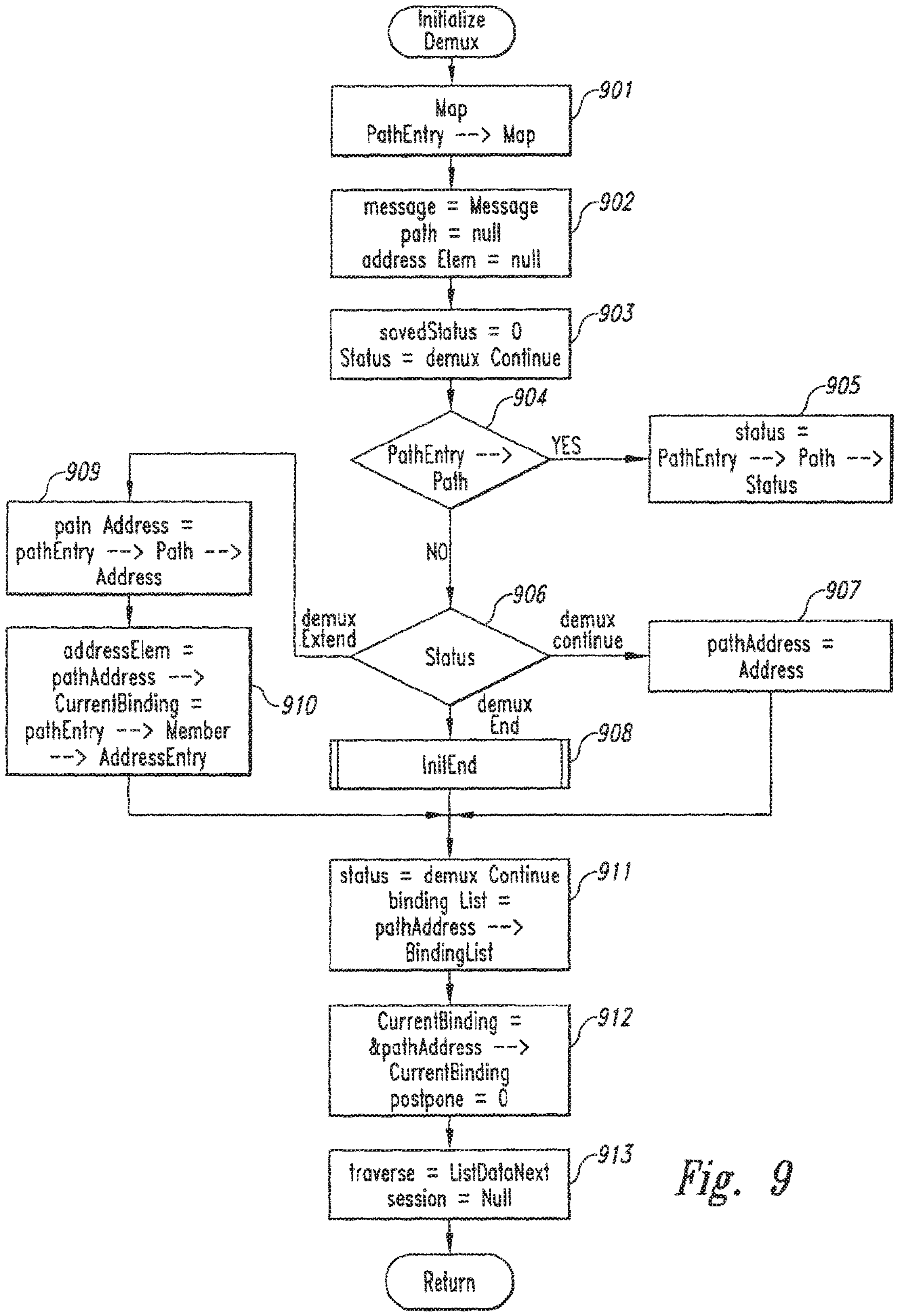

Initialize Demux
901
Map
PathEntry --> Map
902
message = Message
path = null
address Elem = null
903
savedStatus = 0
Status = demux Continue
904
PathEntry --> Path
YES
905
status = PathEntry --> Path --> Status
NO
906
Status
demux Extend
demux continue
demux End
907
pathAddress = Address
909
pain Address = pathEntry --> Path --> Address
910
addressElem = pathAddress --> CurrentBinding = pathEntry --> Member --> AddressEntry
908
InitEnd
911
status = demux Continue
binding List = pathAddress --> BindingList
912
CurrentBinding = &pathAddress --> CurrentBinding
postpone = 0
913
traverse = ListDataNext
session = Null
Return

Fig. 9

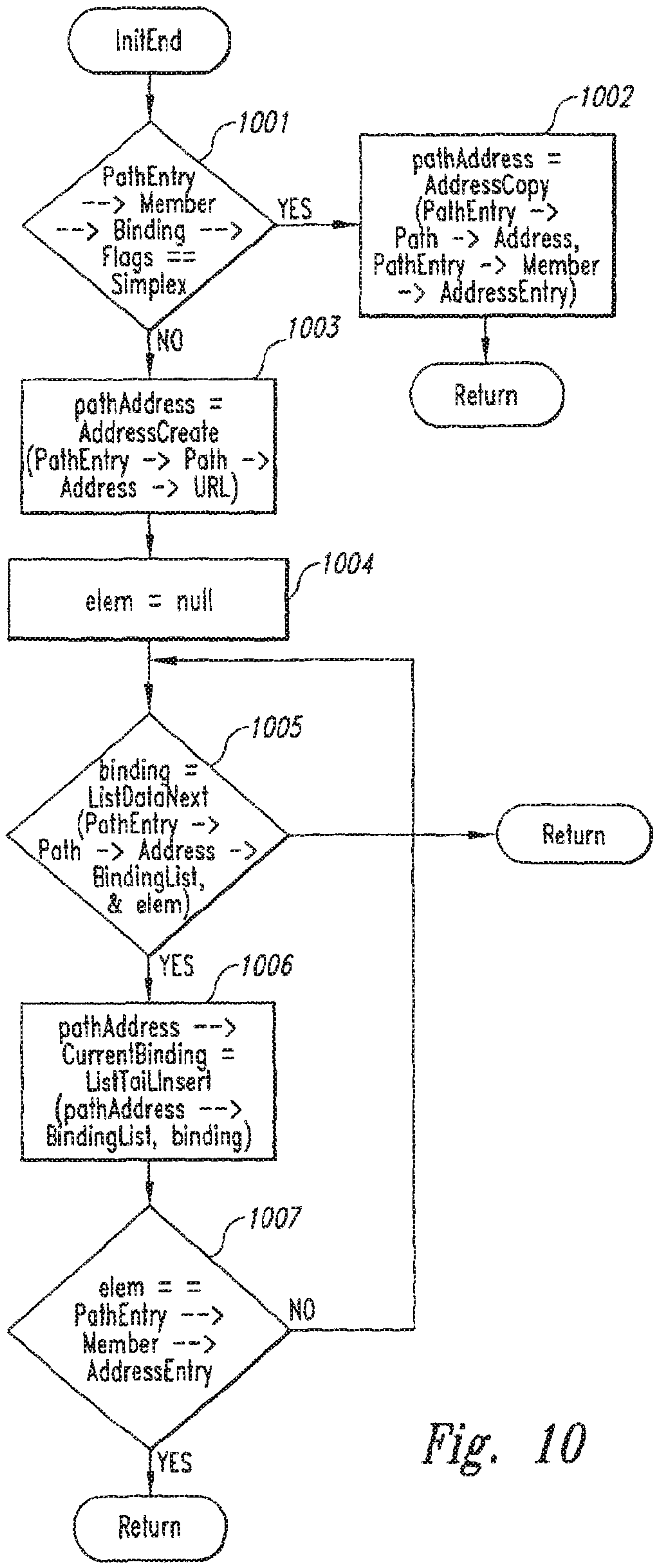

InitEnd
1001
PathEntry --> Member --> Binding --> Flags == Simplex
YES
NO
1002
pathAddress = AddressCopy (PathEntry -> Path -> Address, PathEntry -> Member -> AddressEntry)
Return
1003
pathAddress = AddressCreate (PathEntry -> Path -> Address -> URL)
1004
elem = null
1005
binding = ListDataNext (PathEntry -> Path -> Address -> BindingList, & elem)
Return
YES
1006
pathAddress --> CurrentBinding = ListTailInsert (pathAddress --> BindingList, binding)
1007
elem = = PathEntry --> Member --> AddressEntry
NO
YES
Return

Fig. 10

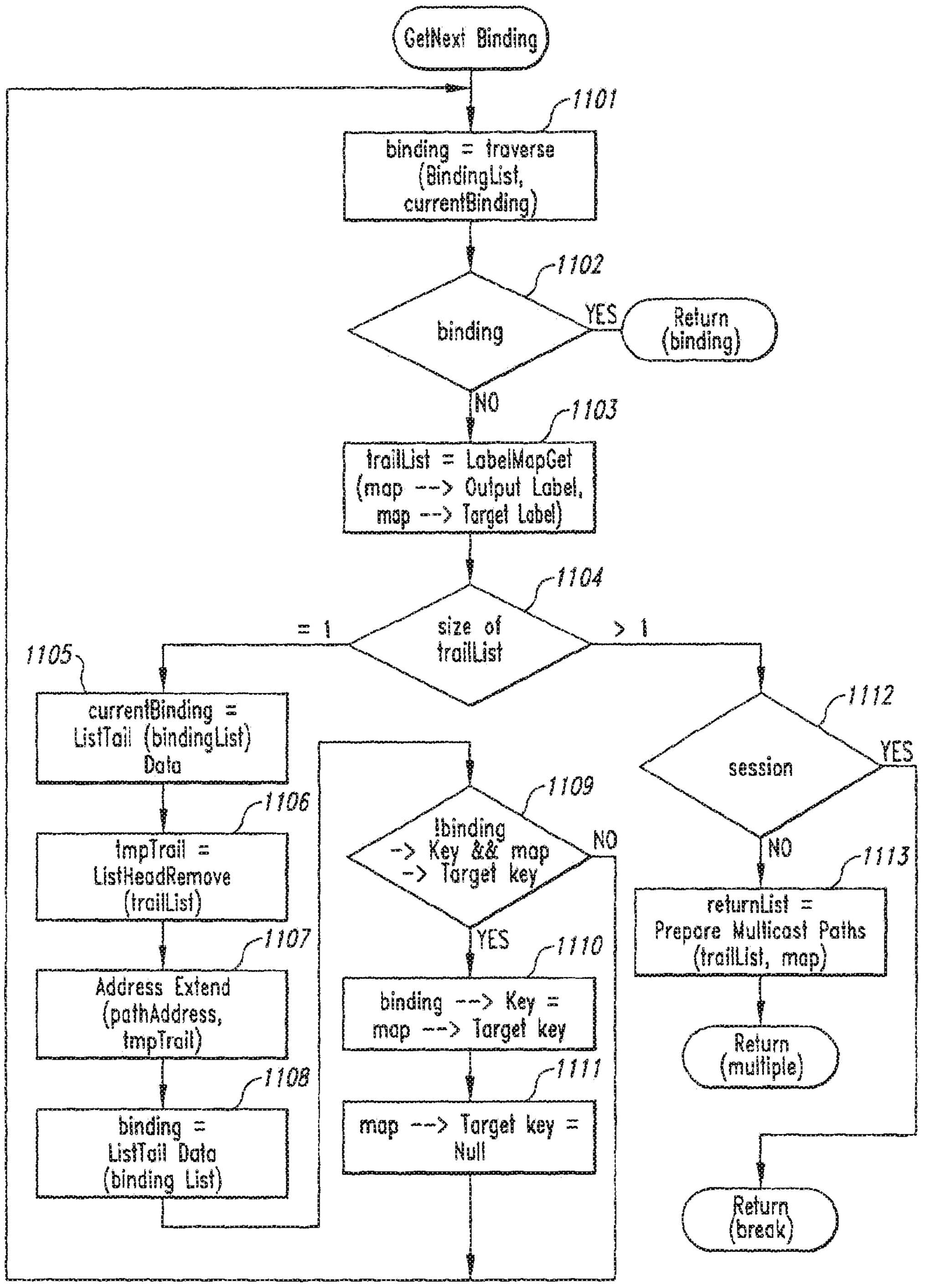

GetNext Binding
1101
binding = traverse (BindingList, currentBinding)
1102
binding
YES
Return (binding)
NO
1103
trailList = LabelMapGet (map --> Output Label, map --> Target Label)
1104
size of trailList
= 1
> 1
1105
currentBinding = ListTail (bindingList) Data
1106
tmpTrail = ListHeadRemove (trailList)
1107
Address Extend (pathAddress, tmpTrail)
1108
binding = ListTail Data (binding List)
1109
!binding -> Key && map -> Target key
NO
YES
1110
binding --> Key = map --> Target key
1111
map --> Target key = Null
1112
session
YES
NO
1113
returnList = Prepare Multicast Paths (trailList, map)
Return (multiple)
Return (break)

Fig. 11

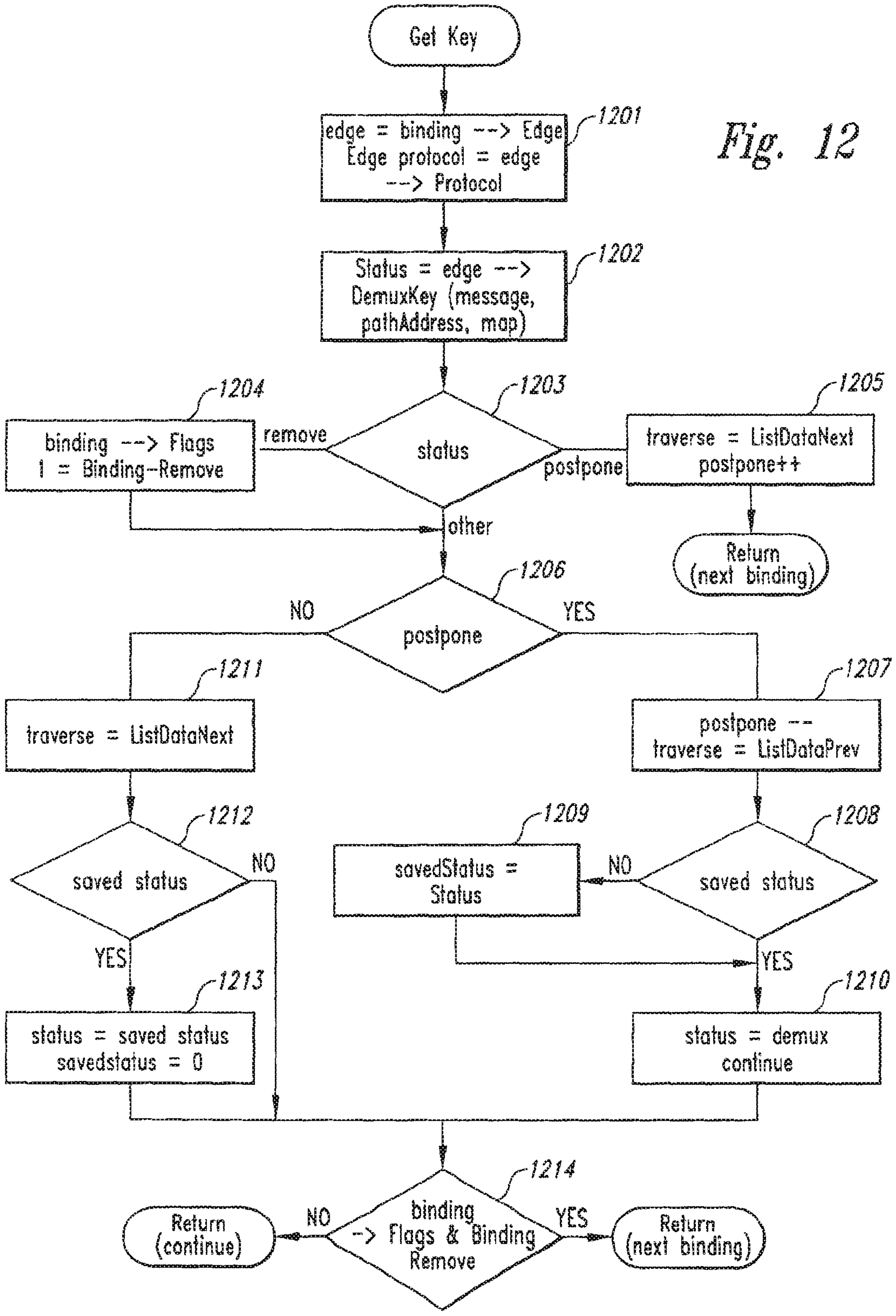

Get Key
edge = binding --> Edge
Edge protocol = edge --> Protocol
1201
Status = edge --> DemuxKey (message, pathAddress, map)
1202
status
1203
remove
binding --> Flags | = Binding-Remove
1204
postpone
traverse = ListDataNext postpone++
1205
Return (next binding)
other
postpone
1206
NO
YES
traverse = ListDataNext
1211
postpone --
traverse = ListDataPrev
1207
saved status
1212
NO
YES
status = saved status
savedstatus = 0
1213
saved status
1208
NO
savedStatus = Status
1209
YES
status = demux continue
1210
binding -> Flags & Binding Remove
1214
NO
Return (continue)
YES
Return (next binding)

Fig. 12

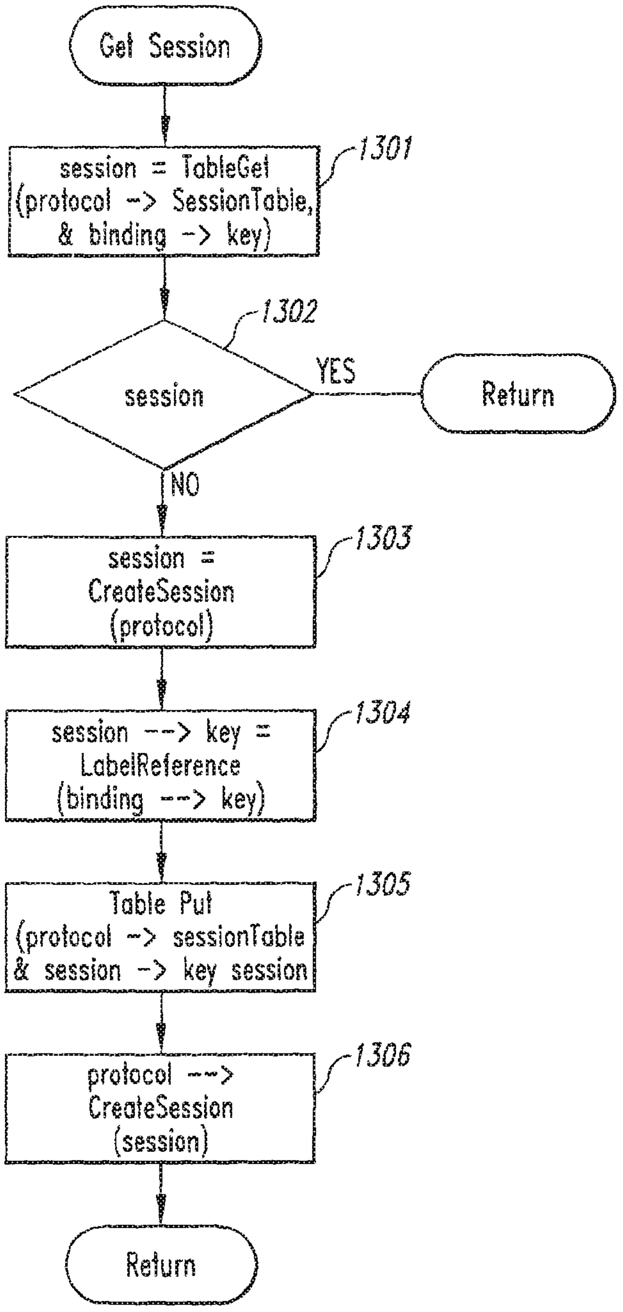

Get Session
session = TableGet (protocol -> SessionTable, & binding -> key)
1301
session
1302
YES
Return
NO
session = CreateSession (protocol)
1303
session --> key = LabelReference (binding --> key)
1304
Table Put (protocol -> sessionTable & session -> key session
1305
protocol --> CreateSession (session)
1306
Return

Fig. 13

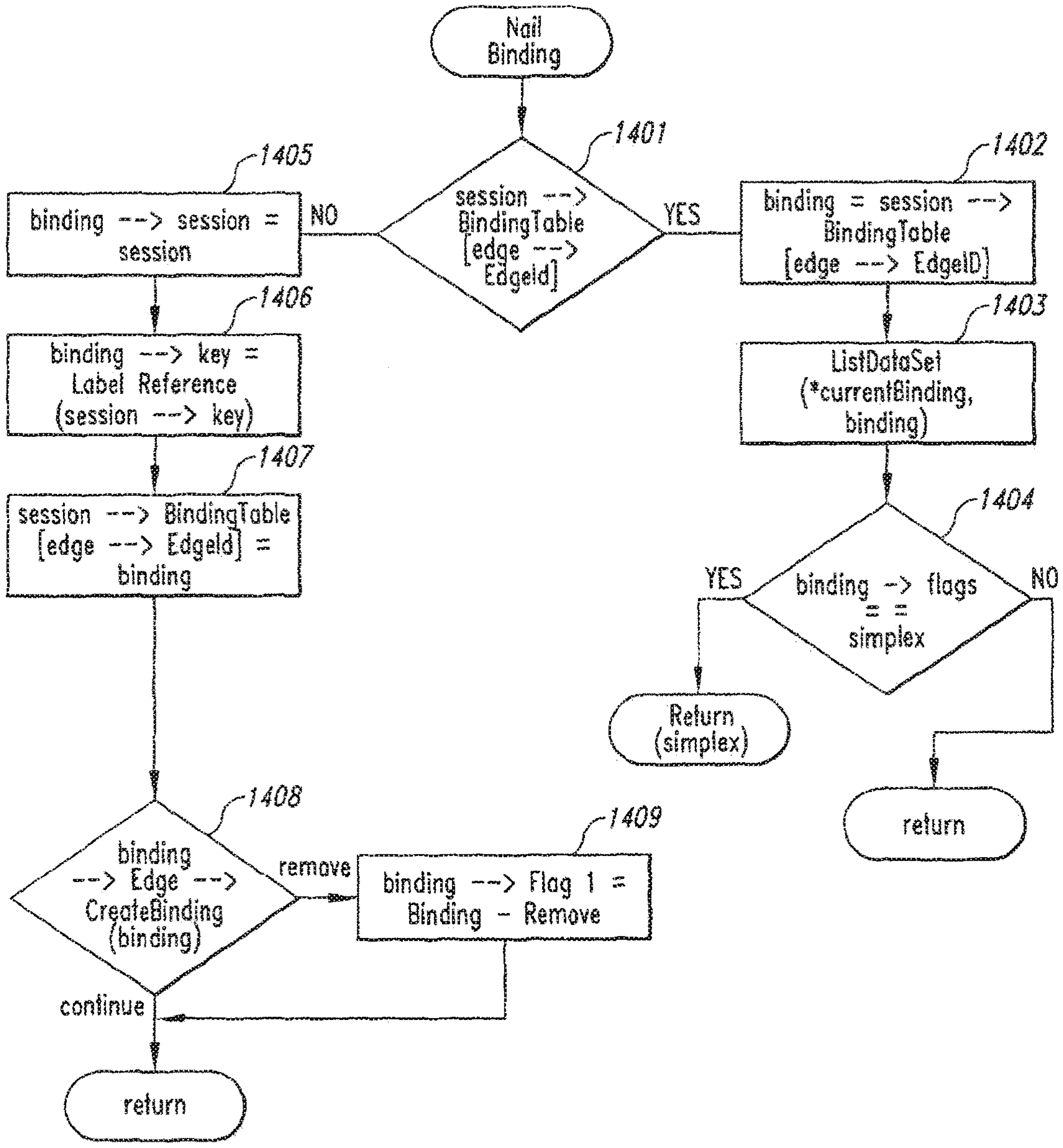

Nail Binding
1401
session --> BindingTable [edge --> EdgeId]
YES
NO
1402
binding = session --> BindingTable [edge --> EdgeID]
1403
ListDataSet (*currentBinding, binding)
1404
binding -> flags = = simplex
YES
NO
Return (simplex)
return
1405
binding --> session = session
1406
binding --> key = Label Reference (session --> key)
1407
session --> BindingTable [edge --> EdgeId] = binding
1408
binding --> Edge --> CreateBinding (binding)
remove
1409
binding --> Flag 1 = Binding - Remove
continue
return

Fig. 14

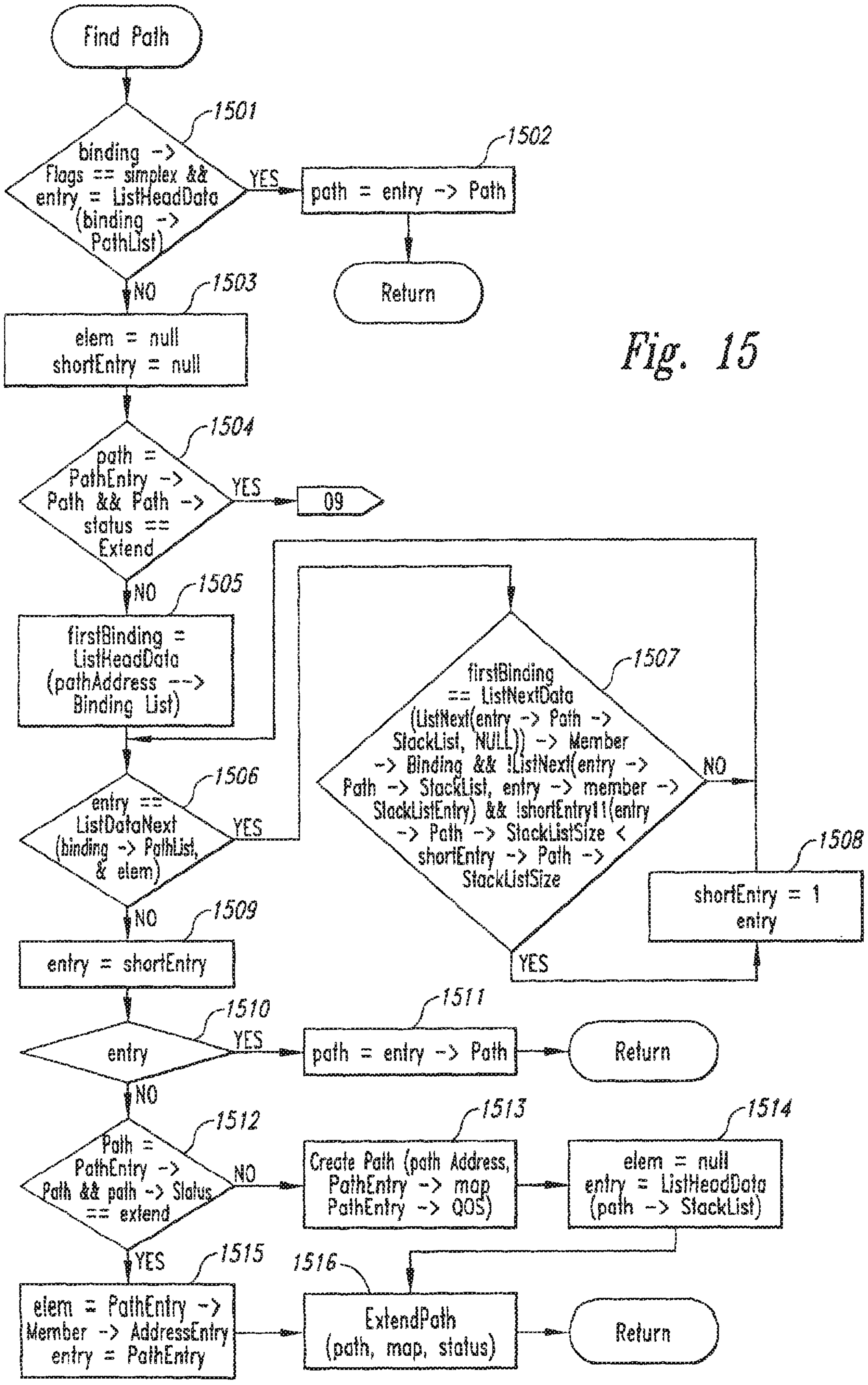

Find Path
1501
binding -> Flags == simplex && entry = ListHeadData (binding -> PathList)
YES
1502
path = entry -> Path
Return
NO
1503
elem = null
shortEntry = null
1504
path = PathEntry -> Path && Path -> status == Extend
YES
09
NO
1505
firstBinding = ListHeadData (pathAddress --> Binding List)
1506
entry == ListDataNext (binding -> PathList, & elem)
YES
1507
firstBinding == ListNextData (ListNext(entry -> Path -> StackList, NULL)) -> Member -> Binding && !ListNext(entry -> Path -> StackList, entry -> member -> StackListEntry) && !shortEntry11(entry -> Path -> StackListSize < shortEntry -> Path -> StackListSize
NO
YES
1508
shortEntry = 1 entry
NO
1509
entry = shortEntry
1510
entry
YES
1511
path = entry -> Path
Return
NO
1512
Path = PathEntry -> Path && path -> Status == extend
NO
1513
Create Path (path Address, PathEntry -> map PathEntry -> QOS)
1514
elem = null
entry = ListHeadData (path -> StackList)
YES
1515
elem = PathEntry -> Member -> AddressEntry
entry = PathEntry
1516
ExtendPath (path, map, status)
Return

Fig. 15

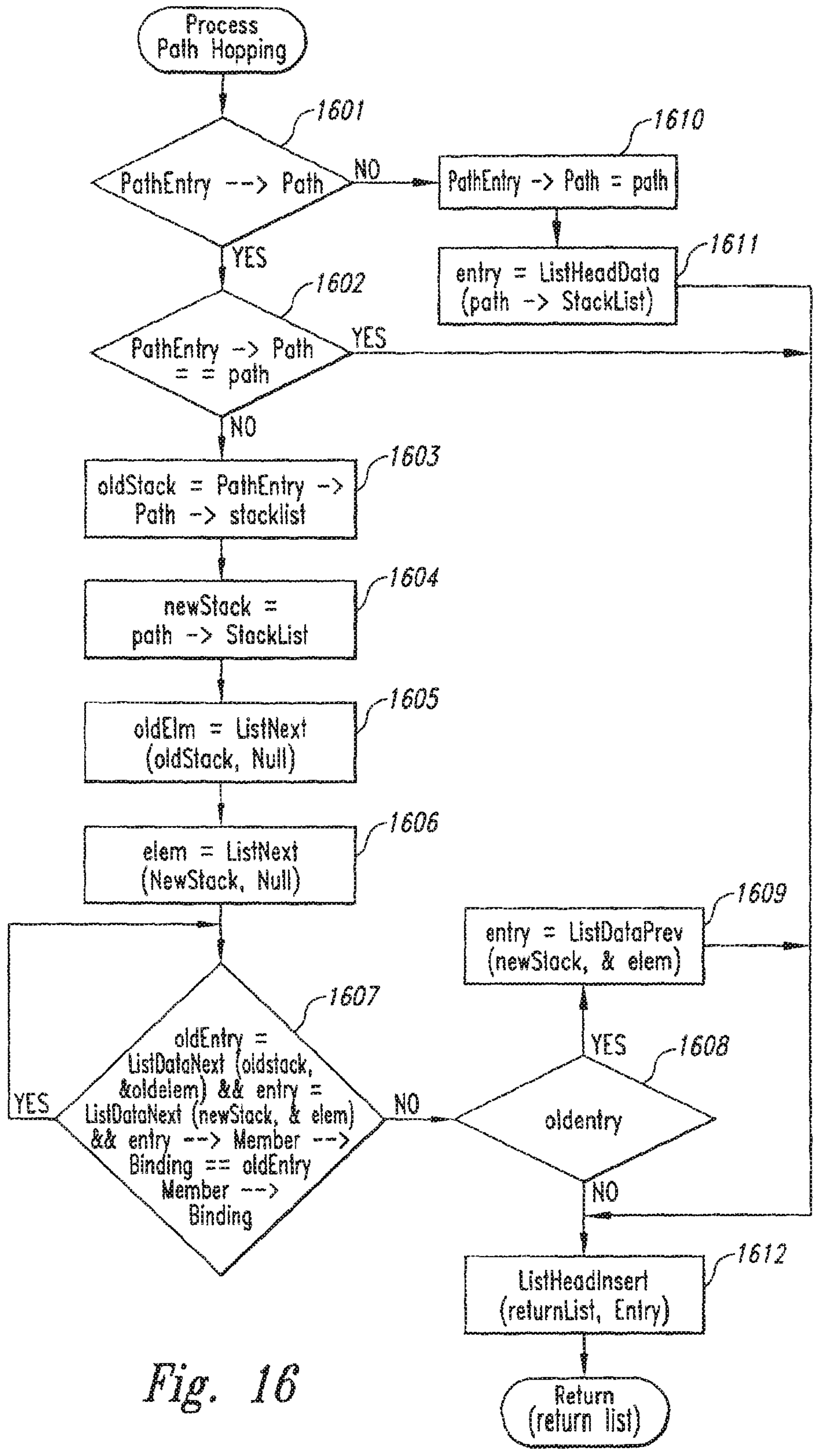

Process Path Hopping
1601
PathEntry --> Path
NO
YES
1610
PathEntry -> Path = path
1611
entry = ListHeadData (path -> StackList)
1602
PathEntry -> Path = = path
YES
NO
1603
oldStack = PathEntry -> Path -> stacklist
1604
newStack = path -> StackList
1605
oldElm = ListNext (oldStack, Null)
1606
elem = ListNext (NewStack, Null)
1609
entry = ListDataPrev (newStack, & elem)
1607
oldEntry = ListDataNext (oldstack, &oldelem) && entry = ListDataNext (newStack, & elem) && entry --> Member --> Binding == oldEntry Member --> Binding
YES
NO
1608
oldentry
YES
NO
1612
ListHeadInsert (returnList, Entry)
Return (return list)

Fig. 16

METHOD AND SYSTEM FOR DATA DEMULTIPLEXING

CROSS REFERENCES TO RELATED APPLICATIONS

The present application is a continuation of U.S. application Ser. No. 15/450,790, filed Mar. 6, 2017, which is a continuation of U.S. application Ser. No. 15/050,027, filed Feb. 22, 2016 (now U.S. Pat. No. 9,591,104), which is a continuation of U.S. application Ser. No. 14/230,952, filed Mar. 31, 2014 (now U.S. Pat. No. 9,270,790), which is a continuation of U.S. application Ser. No. 13/911,324, filed Jun. 6, 2013 (now U.S. Pat. No. 8,694,683), which is a continuation of U.S. application Ser. No. 13/236,090, filed Sep. 19, 2011 (now abandoned), which is a continuation of U.S. application Ser. No. 10/636,314, filed Aug. 6, 2003 (now U.S. Pat. No. 8,055,786), which is a continuation of U.S. application Ser. No. 09/474,664, filed Dec. 29, 1999 (now U.S. Pat. No. 6,629,163); the disclosures of each of the above-referenced applications are incorporated by reference herein in their entireties.

TECHNICAL FIELD

The present invention relates generally to a computer system for data demultiplexing.

BACKGROUND

Computer systems, which are becoming increasingly pervasive, generate data in a wide variety of formats. The Internet is an example of interconnected computer systems that generate data in many different formats. Indeed, when data is generated on one computer system and is transmitted to another computer system to be displayed, the data may be converted in many different intermediate formats before it is eventually displayed. For example, the generating computer system may initially store the data in a bitmap format. To send the data to another computer system, the computer system may first compress the bitmap data and then encrypt the compressed data. The computer system may then convert that compressed data into a TCP format and then into an IP format. The IP formatted data may be converted into a transmission format, such as an ethernet format. The data in the transmission format is then sent to a receiving computer system. The receiving computer system would need to perform each of these conversions in reverse order to convert the data in the bitmap format. In addition, the receiving computer system may need to convert the bitmap data into a format that is appropriate for rendering on output device.

In order to process data in such a wide variety of formats, both sending and receiving computer systems need to have many conversion routines available to support the various formats. These computer systems typically use predefined configuration information to load the correct combination of conversion routines for processing data. These computer systems also use a process-oriented approach when processing data with these conversion routines. When using a process-oriented approach, a computer system may create a separate process for each conversion that needs to take place. A computer system in certain situations, however, can be expected to receive data and to provide data in many different formats that may not be known until the data is received. The overhead of statically providing each possible series of conversion routines is very high. For example, a

computer system that serves as a central controller for data received within a home would be expected to process data received via telephone lines, cable TV lines, and satellite connections in many different formats. The central controller would be expected to output the data to computer displays, television displays, entertainment centers, speakers, recording devices, and so on in many different formats. Moreover, since the various conversion routines may be developed by different organizations, it may not be easy to identify that the output format of one conversion routine is compatible with the input format of another conversion routine.

It would be desirable to have a technique for dynamically identifying a series of conversion routines for processing data. In addition, it would be desirable to have a technique in which the output format of one conversion routine can be identified as being compatible with the input format of another conversion routine. It would also be desirable to store the identification of a series of conversion routines so that the series can be quickly identified when data is received.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a block diagram illustrating example processing of a message by the conversion system.

FIG. **2** is a block diagram illustrating a sequence of edges.

FIG. **3** is a block diagram illustrating components of the conversion system in one embodiment.

FIG. **4** is a block diagram illustrating example path data structures in one embodiment.

FIG. **5** is a block diagram that illustrates the interrelationship of the data structures of a path.

FIG. **6** is a block diagram that illustrates the interrelationship of the data structures associated with a session.

FIGS. **7** A, **7**B, and **7**C comprise a flow diagram illustrating the processing of the message send routine.

FIG. **8** is a flow diagram of the demux routine.

FIG. **9** is a flow diagram of the initialize demux routine.

FIG. **10** is a flow diagram of the init end routine.

FIG. **11** is a flow diagram of a routine to get the next binding.

FIG. **12** is a flow diagram of the get key routine.

FIG. **13** is a flow diagram of the get session routine.

FIG. **14** is a flow diagram of the nail binding routine.

FIG. **15** is a flow diagram of the find path routine.

FIG. **16** is a flow diagram of the process of path hopping routine.

DETAILED DESCRIPTION

A method and system for converting a message that may contain multiple packets from an source format into a target format. When a packet of a message is received, the conversion system in one embodiment searches for and identifies a sequence of conversion routines (or more generally message handlers) for processing the packets of the message by comparing the input and output formats of the conversion routines. (A message is a collection of data that is related in some way, such as stream of video or audio data or an email message.) The identified sequence of conversion routines is used to convert the message from the source format to the target format using various intermediate formats. The conversion system then queues the packet for processing by the identified sequence of conversion routines. The conversion system stores the identified sequence so that the sequence can be quickly found (without searching) when the next packet in the message is received. When subsequent packets of the message are received, the conversion system identifies the sequence and queues the packets for pressing by the sequence. Because the conversion system receives multiple messages with different source and target formats and identifies a sequence of conversion routines for each message, the conversion systems effectively "demultiplexes" the messages. That is, the conversion system demultiplexes the messages by receiving the message, identifying the sequence of conversion routines, and controlling the processing of each message by the identified sequence. Moreover, since the conversion routines may need to retain state information between the receipt of one packet of a message and the next packet of that message, the conversion system maintains state information as an instance or session of the conversion routine. The conversion system routes all packets for a message through the same session of each conversion routine so that the same state or instance information can be used by all packets of the message. A sequence of sessions of conversion routines is referred to as a "path." In one embodiment, each path has a path thread associated with it for processing of each packet destined for that path.

In one embodiment, the packets of the messages are initially received by "drivers," such as an Ethernet driver. When a driver receives a packet, it forwards the packet to a forwarding component of the conversion system. The forwarding component is responsible for identifying the session of the conversion routine that should next process the packet and invoking that conversion routine. When invoked by a driver, the forwarding component may use a demultiplexing ("demux") component to identify the session of the first conversion routine of the path that is to process the packet and then queues the packet for processing by the path. A path thread is associated with each path. Each path thread is responsible for retrieving packets from the queue of its path and forwarding the packets to the forwarding component. When the forwarding component is invoked by a path thread, it initially invokes the first conversion routine in the path. That conversion routine processes the packet and forwards the processed packet to the forwarding component, which then invokes the second conversion routine in the path. The process of invoking the conversion routines and forwarding the processed packet to the next conversion routine continues until the last conversion routine in the path is invoked. A conversion routine may defer invocation of the forwarding component until it aggregates multiple packets or may invoke the forwarding component multiple times for a packet once for each sub-packet.

The forwarding component identifies the next conversion routine in the path using the demux component and stores that identification so that the forwarding component can quickly identify the conversion routine when subsequent packets of the same message are received. The demux component, searches for the conversion routine and session that is to next process a packet. The demux component then stores the identification of the session and conversion routine as part of a path data structure so that the conversion system does not need to search for the session and conversion routine when requested to demultiplex subsequent packets of the same message. When searching for the next conversion routine, the demux component invokes a label map get component that identifies the next conversion routine. Once the conversion routine is found, the demux component identifies the session associated with that message by, in one embodiment, invoking code associated with the conversion routine. In general, the code of the conversion routine determines what session should be associated with a message. In certain situations, multiple messages may

share the same session. The demux component then extends the path for processing that packet to include that session and conversion routine. The sessions are identified so that each packet is associated with the appropriate state information. The dynamic identification of conversion routines is described in U.S. patent application Ser. No. 11/933,093, filed on Oct. 31, 2007 (now U.S. Pat. No. 7,730,211), entitled "Method and System for Generating a Mapping Between Types of Data," which is hereby incorporated by reference.

FIG. **1** is a block diagram illustrating example processing of a message by the conversion system. The driver **101** receives the packets of the message from a network. The driver performs any appropriate processing of the packet and invokes a message send routine passing the processed packet along with a reference path entry **150**. The message send routine is an embodiment of the forwarding component. A path is represented by a series of path entries, which are represented by triangles. Each member path entry represents a session and conversion routine of the path, and a reference path entry represents the overall path. The passed reference path entry **150** indicates to the message send routine that it is being invoked by a driver. The message send routine invokes the demux routine **102** to search for and identify the path of sessions that is to process the packet. The demux routine may in turn invoke the label map get routine **104** to identify a sequence of conversion routines for processing the packet. In this example, the label map get routine identifies the first three conversion routines, and the demux routine creates the member path entries **151**, **152**, **153** of the path for these conversion routines. Each path entry identifies a session for a conversion routine, and the sequence of path entries **151**-**155** identifies a path. The message send routine then queues the packet on the queue **149** for the path that is to process the packets of the message. The path thread **105** for the path retrieves the packet from the queue and invokes the message send routine **106** passing the packet and an indication of the path. The message send routine determines that the next session and conversion routine as indicated by path entry **151** has already been found. The message send routine then invokes the instance of the conversion routine for the session. The conversion routine processes the packet and then invokes the message send routine **107**. This processing continues until the message send routine invokes the demux routine **110** after the packet is processed by the conversion routine represented by path entry **153**. The demux routine examines the path and determines that it has no more path entries. The demux routine then invokes the label map get routine **111** to identify the conversion routines for further processing of the packet. When the conversion routines are identified, the demux routine adds path entries **154**, **155** to the path. The messages send routine invokes the conversion routine associated with path entry **154**. Eventually, the conversion routine associated with path entry **155** performs the final processing for the path.

The label map get routine identifies a sequence of "edges" for converting data in one format into another format. Each edge corresponds to a conversion routine for converting data from one format to another. Each edge is part of a "protocol" (or more generally a component) that may include multiple related edges. For example, a protocol may have edges that each convert data in one format into several different formats. Each edge has an input format and an output format. The label map get routine identifies a sequence of edges such that the output format of each edge is compatible with the input format of another edge in the sequence, except for the input format of the first edge in the sequence and the output format of the last edge in the sequence. FIG. **2** is a block diagram illustrating a sequence of edges. Protocol PI includes an edge for converting format D**1** to format D**2** and an edge for converting format D**1** to format D**3**; protocol P**2** includes an edge for converting format D**2** to format D**5**, and so on. A 30 sequence for converting format D **1** to format D **15** is shown by the curved lines and is defined by the address "P**1**:I, P**2**:**1**, P**3**:**2**, P**4**:**7**." When a packet of data in format D I is processed by this sequence, it is converted to format DIS. During the process, the packet of data is sequentially converted to format D**2**, D**5**, and D**13**. The output format of protocol P**2**, edge **1** (i.e., P**2**:**1**) is format D**5**, but the input format of P**3**:**2** is format D**10**. The label map get routine uses an aliasing mechanism by which two formats, such as D**5** and D**10** are identified as being compatible. The use of aliasing allows different names of the same format or compatible formats to be correlated.

FIG. **3** is a block diagram illustrating components of the conversion system in one embodiment. The conversion system **300** can operate on a computer system with a central processing unit **301**, I/O devices **302**, and memory **303**. The **110** devices may include an Internet connection, a connection to various output devices such as a television, and a connection to various input devices such as a television receiver. The media mapping system may be stored as instructions on a computer-readable medium, such as a disk drive, memory, or data transmission medium. The data structures of the media mapping system may also be stored on a computer-readable medium. The conversion system includes drivers **304**, a•forwarding component **305**, a demux component **306**, a label map get component **307**, path data structures **308**, conversion routines **309**, and instance data **310**. Each driver receives data in a source format and forwards the data to the forwarding component. The forwarding component identifies the next conversion routine in the path and invokes that conversion routine to process a packet. The forwarding component may invoke the demux component to search for the next conversion routine and add that conversion routine to the path. The demux component may invoke the label map get component to identify the next conversion routine to process the packet. The demux component stores information defining the paths in the path structures. The conversion routines store their state information in the instance data.

FIG. **4** is a block diagram illustrating example path data structures in one embodiment. The demux component identifies a sequence of "edges" for converting data in one format into another format by invoking the label map get component. Each edge corresponds to a conversion routine for converting data from one format to another. As discussed above, each edge is part of a "protocol" that may include multiple related edges. For example, a protocol may have edges that each convert data in one format into several different formats. Each edge has as an input format ("input label") and an output format ("output label"). Each rectangle represents a session **410**, **420**, **430**, **440**, **450** for a protocol. A session corresponds to an instance of a protocol. That is, the session includes the protocol and state information associated with that instance of the protocol. Session **410** corresponds to a session for an Ethernet protocol; session **420** corresponds to a session for an IP protocol; and sessions **430**, **440**, **450** correspond to sessions for a TCP protocol. FIG. **4** illustrates three paths **461**, **462**, **463**. Each path includes edges **411**, **421**, **431**. The paths share the same Ethernet session **410** and IP session **420**, but each path has a unique TCP session **430**, **440**, **450**. Thus, path **461** includes sessions **410**, **420**, and **430**; path **462** includes sessions **410**,

420, and **440**; and path **463** includes sessions **410**, **420**, and **450**. The conversion system represents each path by a sequence of path entry structures. Each path entry structure is represented by a triangle. Thus, path **461** is represented by path entries **415**, **425**, and **433**. The conversion system represents the path entries of a path by a stack list. Each path also has a queue **471**, **472**, **473** associated with it. Each queue stores the messages that are to be processed by the conversion routines of the edges of the path. Each session includes a binding **412**, **422**, **432**, **442**, **452** that is represented by an oblong shape adjacent to the corresponding edge. A binding for an edge of a session represents those paths that include the edge. The binding **412** indicates that three paths are bound (or "nailed") to edge **411** of the Ethernet session **410**. The conversion system uses a path list to track the paths that are bound to a binding. The path list of binding **412** identifies path entries **413**, **414**, and **415**.

FIG. **5** is a block diagram that illustrates the interrelationship of the data structures of a path. Each path has a corresponding path structure **501** that contains status information and pointers to a message queue structure **502**, a stack list structure **503**, and a path address structure **504**. The status of a path can be extend, continue, or end. Each message handler returns a status for the path. The status of extend means that additional path entries should be added to the path. The status of end means that this path should end at this point and subsequent processing should continue at a new path. The status of continue means that the protocol does not care how the path is handled. In one embodiment, when a path has a status of continue, the system creates a copy of the path and extends the copy. The message queue structure identifies the messages (or packets of a message) that are queued up for processing by the path and identifies the path entry at where the processing should start. The stack list structure contains a list of pointers to the path entry structures **505** that comprise the path. Each path entry structure contains a pointer to the corresponding path data structure, a pointer to a map structure **507**, a pointer to a multiplex list **508**, a pointer to the corresponding path address structure, and a pointer to a member structure **509**. A map structure identifies the output label of the edge of the path entry and optionally a target label and a target key. A target key identifies the session associated with the protocol that converts the packet to the target label. (The terms "media," "label," and "format" are used interchangeably to refer to the output of a protocol.) The multiplex list is used during the demux process to track possible next edges when a path is being identified as having more than one next edge. The member structure indicates that the path entry represents an edge of a path and contains a pointer to a binding structure to which the path entry is associated (or "nailed"), a stack list entry is the position of the path entry within the associated stack list, a path list entry is the position of the path entry within the associated path list of a binding and an address entry is the position of the binding within the associated path address. A path address of a path identifies the bindings to which the path entries are bound. The path address structure contains a URL for the path, the name of the path identified by the address, a pointer to a binding list structure **506**, and the identification of the current binding within the binding list. The URL (e.g., "protocol://tcp(0)/ip (0)/eth(0)") identifies conversion routines (e.g., protocols and edges) of a path in a human-readable format. The URL (universal resource locator) includes a type field (e.g., "protocol") followed by a sequence of items (e.g., "tcp(0)"). The type field specifies the format of the following information in the URL, that specifies that the type field is followed by a sequence of items. Each item identifies a protocol and an edge (e.g., the protocol is "tcp" and the edge is "0"). In one embodiment, the items of a URL may also contain an identifier of state information that is to be used when processing a message. These URLs can be used to illustrate to a user various paths that are available for processing a message. The current binding is the last binding in the path as the path is being built. The binding list structure contains a list of pointers to the binding structures associated with the path. Each binding structure **510** contains a pointer to a session structure, a pointer to an edge structure, a key, a path list structure, and a list of active paths through the binding. The key identifies the state information for a session of a protocol. A path list structure contains pointers to the path entry structures associated with the binding.

FIG. **6** is a block diagram that illustrates the interrelationship of the data structures associated with a session. A session structure **601** contains the context for the session, a pointer to a protocol structure for the session, a pointer to a binding table structure **602** for the bindings associated with the session, and the key. The binding table structure contains a list of pointers to the binding structures **510** for the session. The binding structure is described above with reference to FIG. **5**. The path list structure **603** of the binding structure contains a list of pointers to path entry structures **505**. The path entry structures are described with reference to FIG. **5**.

FIGS. **7** A, **7**B, and **7**C comprise a flow diagram illustrating the processing of the message send routine. The message send routine is passed a message along with the path entry associated with the session that last processed the message. The message send routine invokes the message handler of the next edge in the path or queues the message for processing by a path. The message handler invokes the demux routine to identify the next path entry of the path. When a driver receives a message, it invokes the message send routine passing a reference path entry. The message send routine examines the passed path entry to determine (1) whether multiple paths branch from the path of the passed path entry, (2) whether the passed path entry is a reference with an associated path, or (3) whether the passed path entry is a member with a next path entry. If multiple paths branch from the path of the passed path entry, then the routine recursively invokes the message send routine for each path. If the path entry is a reference with an associated path, then the driver previously invoked the message send routine, which associated a path with the reference path entry, and the routine places the message on the queue for the path. If the passed path entry is a member with a next path entry, then the routine invokes the message handler (i.e., conversion routine of the edge) associated with the next path entry. If the passed path entry is a reference without an associated path or is a member without a next path entry, then the routine invokes the demux routine to identify the next path entry. The routine then recursively invokes the messages send routine passing that next path entry. In decision block **701**, if the passed path entry has a multiplex list, then the path branches off into multiple paths and the routine continues at block **709**, else the routine continues at block **702**. A packet may be processed by several different paths. For example, if a certain message is directed to two different output devices, then the message is processed by two different paths. Also, a message may need to be processed by multiple partial paths when searching for a complete path. In decision block **702**, if the passed path entry is a member, then either the next path entry indicates a nailed binding or the path needs to be extended and the routine continues at block **704**, else the routine continues at block **703**. A nailed binding is a binding (e.g., edge and protocol) is associated with a session. In decision block **703**, the passed path entry is a reference and if the passed path entry has an associated path, then the routine can queue the message for the associated path and the routine continues at block **703**A, else the routine needs to identify a path and the routine continues at block **707**. In block **703**A, the routine sets the entry to the first path entry in the path and continues at block **717**. In block **704**, the routine sets the variable position to the stack list entry of the passed path entry. In decision block **705**, the routine sets the variable next entry to the next path entry in the path. If there is a next entry in the path, then the next session and edge of the protocol have been identified and the routine continues at block **706**, else the routine continues at block **707**. In block **706**, the routine passes the message to the message handler of the edge associated with the next entry and then returns. In block **706**, the routine invokes the demux routine passing the passed message, the address of the passed path entry, and the passed path entry. The demux routine returns a list of candidate paths for processing of the message. In decision block **708**, if at least one candidate path is returned, then the routine continues at block **709**, else the routine returns.

Blocks **709**-**716** illustrate the processing of a list of candidate paths that extend from the passed path entry. In blocks **710**-**716**, the routine loops selecting each candidate path and sending the message to be process by each candidate path. In block **710**, the routine sets the next entry to the first path entry of the next candidate path. In decision block **711**, if all the candidate paths have not yet been processed, then the routine continues at block **712**, else the routine returns. In decision block **712**, if the next entry is equal to the passed path entry, then the path is to be extended and the routine continues at block **705**, else the routine continues at block **713**. The candidate paths include a first path entry that is a reference path entry for new paths or that is the last path entry of a path being extended. In decision block **713**, if the number of candidate paths is greater than one, then the routine continues at block **714**, else the routine continues at block **718**. In decision block **714**, if the passed path entry has a multiplex list associated with it, then the routine continues at block **716**, else the routine continues at block **715**. In block **715**, **11** the routine associates the list of candidate path with the multiplex list of the passed path entry and continues at block **716**. In block **716**, the routine sends the message to the next entry by recursively invoking the message send routine. The routine then loops to block **710** to select the next entry associated with the next candidate path.

Blocks **717**-**718** are performed when the passed path entry is a reference path entry that has a path associated with it. In block **717**, if there is a path associated with the next entry, then the routine continues at block **718**, else the routine returns. In block **718**, the routine queues the message for the path of the next entry and then returns.

FIG. **8** is a flow diagram of the demux routine. This routine is passed the packet (message) that is received, an address structure, and a path entry structure. The demux routine extends a path, creating one if necessary. The routine loops identifying the next binding (edge and protocol) that is to process the message and "nailing" the binding to a session for the message, if not already nailed. After identifying the nailed binding, the routine searches for the shortest path through the nailed binding, creating a path if none exists. In block **801**, the routine invokes the initialize demux routine. In blocks **802**-**810**, the routine loops identifying a path or portion of a path for processing the passed message. In decision block **802**, if there is a current status, which was returned by the demux key routine that was last invoked (e.g., continue, extend, end, or postpone), then the routine continues at block **803**, else the routine continues at block **811**. In block **803**, the routine invokes the get next binding routine. The get next binding routine returns the next binding in the path. The binding is the edge of a protocol. That routine extends the path as appropriate to include the binding. The routine returns a return status of break, binding, or multiple. The return status of binding indicates that the next binding in the path was found by extending the path as appropriate and the routine continues to "nail" the binding to a session as appropriate. The return status of multiple means that multiple trails (e.g., candidate paths) were identified as possible extensions of the path. In a decision block **804**, if the return status is break, then the routine continues at block **811**. If the return status is multiple, then the routine returns. If the return status is binding, then the routine continues at block **805**. In decision block **805**, if the retrieved binding is nailed as indicated by being assigned to a session, then the routine loops to block **802**, else the routine continues at block **806**. In block **806**, the routine invokes the get key routine of the edge associated with the binding. The get key routine creates the key for the session associated with the message. If a key cannot be created until subsequent bindings are processed or because the current binding is to be removed, then the get key routine returns a next binding status, else it returns a continue status. In decision block **807**, if the return status of the get key routine is next binding, then the routine loops to block **802** to get the next binding, else the routine continues at block **808**. In block **808**, the routine invokes the routine get session. The routine get session returns the session associated with the key, creating a new session if necessary. In block **809**, the routine invokes the routine nail binding. The routine nail binding retrieves the binding if one is already nailed to the session. Otherwise, that routine nails the binding to the session. In decision block **810**, if the nail binding routine returns a status of simplex, then the routine continues at block **811** because only one path can use the session, else the routine loops to block **802**. Immediately upon return from the nail binding routine, the routine may invoke a set map routine of the edge passing the session and a map to allow the edge to set its map. In block **811**, the routine invokes the find path routine, which finds the shortest path through the binding list and creates a path if necessary. In block **812**, the routine invokes the process path hopping routine, which determines whether the identified path is part of a different path. Path hopping occurs when, for example, IP fragments are built up along separate paths, but once the fragments are built up they can be processed by the same subsequent path.

FIG. **9** is a flow diagram of the initialize demux routine. This routine is invoked to initialize the local data structures that are used in the demux process and to identify the initial binding. The demux routine finds the shortest path from the initial binding to the final binding. If the current status is demux extend, then the routine is to extend the path of the passed path entry by adding additional path entries. If the current status is demux end, then the demux routine is ending the current path. If the current status is demux continue, then the demux routine is in the process of continuing to extend or in the process of starting a path identified by the passed address. In block **901**, the routine sets the local map structure to the map structure in the passed path entry structure. The map structure identifies the output label, the target label, and the target key. In the block **902**, the routine initializes the local message structure to the passed message structure and initializes the pointers path

and address element to null. In block **903**, the routine sets of the variable saved status to 0 and the variable status to demux continue. The variable saved status is used to track the status of the demux process when backtracking to nail a binding whose nail was postponed. In decision block **904**, if the passed path entry is associated with a path, then the routine continues at block **905**, else the routine continues at block **906**. In block **905**, the routine sets the variable status to the status of that path. In block **906**, if the variable status is demux continue, then the routine continues at block **907**. If the variable status is demux end, then the routine continues at block **908**. If the variable status is demux extend, then the routine continues at block **909**. In block **907**, the status is demux continue, and the routine sets the local pointer path address to the passed address and continues at block **911**. In block **908**, the status is demux end, and the routine invokes the init end routine and continues at block **911**. In block **909**, the status is demux extend, and the routine sets the local path address to the address of the path that contains the passed path entry. In block **910**, the routine sets the address element and the current binding of the path address pointed to by the local pointer path address to the address entry of the member structure of the passed path entry. In the block **911**, the routine sets the local variable status to demux continue and sets the local binding list structure to the binding list structure from the local path address structure. In block **912**, the routine sets the local pointer current binding to the address of the current binding pointed to by local pointer path address and sets the local variable postpone to 0. In block **913**, the routine sets the function traverse to the function that retrieves the next data in a list and sets the local pointer session to null. The routine then returns.

FIG. **10** is a flow diagram of the init end routine. If the path is simplex, then the routine creates a new path from where the other one ended, else the routine creates a copy of the path. In block **1001**, if the binding of the passed path entry is simplex (i.e., only one path can be bound to this binding), then the routine continues at block **1002**, else the routine continues at block **1003**. In block **1002**, the routine sets the local pointer path address to point to an address structure that is a copy of the address structure associated with the passed path entry structure with its current binding to the address entry associated with the passed path entry structure, and then returns. In block **1003**, the routine sets the local pointer path address to point to an address structure that contains the URL of the path that contains the passed path entry. In block **1004**, the routine sets the local pointer element to null to initialize the selection of the bindings. In blocks **1005** through **1007**, the routine loops adding all the bindings for the address of the passed path entry that include and are before the passed path entry to the address pointed to by the local path address. In block **1005**, the routine retrieves the next binding from the binding list starting with the first. If there is no such binding, then the routine returns, else the routine continues at block **1006**. In block **1006**, the routine adds the binding to the binding list of the local path address structure and sets the current binding of the local variable path address. In the block **1007**, if the local pointer element is equal to the address entry of the passed path entry, then the routine returns, else the routine loops to block **1005** to select the next binding.

FIG. **11** is a flow diagram of a routine to get the next binding. This routine returns the next binding from the local binding list. If there is no next binding, then the routine invokes the routine label map get to identify the list of edges ("trails") that will map the output label to the target label. If only one trail is identified, then the binding list of path address is extended by the edges of the trail. If multiple trails are identified, then a path is created for each trail and the routine returns so that the demux process can be invoked for each created path. In block **1101**, the routine sets the local pointer binding to point to the next or previous (as indicated by the traverse function) binding in the local binding list. In block **1102**, if a binding was found, then the routine returns an indication that a binding was found, else the routine continues at block **1103**. In block **1103**, the routine invokes the label map get function passing the output label and target label of the local map structure. The label map get function returns a trail list. A trail is a list of edges from the output label to the target label. In decision block **1104**, if the size of the trail list is one, then the routine continues at block **1105**, else the routine continues at block **1112**. In blocks **1105-1111**, the routine extends the binding list by adding a binding data structure for each edge in the trail. The routine then sets the local binding to the last binding in the binding list. In block **1108**, the routine sets the local pointer current binding to point to the last binding in the local binding list. In block **1106**, the routine sets the local variable temp trail to the trail in the trail list. In block **1107**, the routine extends the binding list by temp trail by adding a binding for each edge in the trail. These bindings are not yet nailed. In block **1108**, the routine sets the local binding to point to the last binding in the local binding list. In decision block **1109**, if the local binding does not have a key for a session and the local map has a target key for a session, then the routine sets the key for the binding to the target key of the local map and continues at block **1110**, else the routine loops to block **1101** to retrieve the next binding in path. In block **1110**, the routine sets the key of the local binding to the target key of the local map. In block **1111**, the routine sets the target key of the local map to null and then loop to block **1101** to return the next binding. In decision block **1112**, if the local session is set, then the demultiplexing is already in progress and the routine returns a break status. In block **1113**, the routine invokes a prepare multicast paths routine to prepare a path entry for each trail in the trail list. The routine then returns a multiple status.

FIG. **12** is a flow diagram of the get key routine. The get key routine invokes an edge's demux key routine to retrieve a key for the session associated with the message. The key identifies the session of a protocol. The demux key routine creates the appropriate key for the message. The demux key routine returns a status of remove, postpone, or other. The status of remove indicates that the current binding should be removed from the path. The status of postpone indicates that the demux key routine cannot create the key because it needs information provided by subsequent protocols in the path. For example, a TCP session is defined by a combination of a remote and local port address and an IP address. Thus, the TCP protocol postpones the creating of a key until the IP protocol identifies the IP address. The get key routine returns a next binding status to continue at the next binding in the path. Otherwise, the routine returns a continue status. In block **1201**, the routine sets the local edge to the edge of the local binding (current binding) and sets the local protocol to the protocol of the local edge. In block **1202**, the routine invokes the demux key routine of the local edge passing the local message, local path address, and local map. The demux key routine sets the key in the local binding. In decision block **1203**, if the demux key routine returns a status of remove, then the routine continues at block **1204**. If the demux key routine returns a status of postpone, then the routine continues at block **1205**, else the routine continues at block **1206**. In block **1204**, the routine sets the flag of the local binding to indicate that the binding is to be removed and continues at block **1206**. In block **1205**, the routine sets the variable traverse to the function to list the next data, increments the variable postpone, and then returns a next binding status. In blocks **1206-1214**, the routine processes the postponing of the creating of a key. In blocks **1207-1210**, if the creating of a key has been postponed, then the routine indicates to backtrack on the path, save the demux status, and set the demux status to demux continue. In blocks **1211-1213**, if the creating of a key has not been postponed, then the routine indicates to continue forward in the path and to restore any saved demux status. The save demux status is the status associated by the binding where the backtrack started. In decision block **1206**, if the variable postpone is set, then the routine continues at block **1207**, else the routine continues at block **1211**. In block **1207**, the routine decrements the variable postpone and sets the variable traverse to the list previous data function. In decision block **1208**, if the variable saved status is set, then the routine continues at block **1210**, else the routine continues at block **1209**. The variable saved status contains the status of the demux process when the demux process started to backtrack. In block **1209**, the routine sets the variable saved status to the variable status. In block **1210**, the routine sets the variable status to demux continue and continues at block **1214**. In block **1211**, the routine sets the variable traverse to the list next data function. In decision block **1212**, if the variable saved status in set, then the routine continues at block **1213**, else the routine continues at block **1214**. In block **1213**, the routine sets the variable status to the variable saved status and sets the variable saved status to 0. In decision block **1214**, if the local binding indicates that it is to be removed, then the routine returns a next binding status, else the routine returns a continue status.

FIG. **13** is a flow diagram of the get session routine. This routine retrieves the session data structure, creating a data structure session if necessary, for the key indicated by the binding. In block **1301**, the routine retrieves the session from the session table of the local protocol indicated by the key of the local binding. Each protocol maintains a mapping from each key to the session associated with the key. In decision block **1302**, if there is no session, then the routine continues at block **1303**, else the routine returns. In block **1303**, the routine creates a session for the local protocol. In block **1304**, the routine initializes the key for the local session based on the key of the local binding. In block **1305**, the routine puts the session into the session table of the local protocol. In block **1306**, the routine invokes the create session function of the protocol to allow the protocol to initialize its context and then returns.

FIG. **14** is a flow diagram of the nail binding routine. This routine determines whether a binding is already associated with ("nailed to") the session. If so, the routine returns that binding. If not, the routine associates the binding with the session. The routine returns a status of simplex to indicate that only one path can extend through the nailed binding. In decision block **1401**, if the binding table of the session contains an entry for the edge, then the routine continues at block **1402**, else the routine continues at block **1405**. In block **1402**, the routine sets the binding to the entry from the binding table of the local session for the edge. In block **1403**, the routine sets the current binding to point to the binding from the session. In block **1404**, if the binding is simplex, then the routine returns a simplex status, else the routine returns. Blocks **1405** through **1410** are performed when there is no binding in the session for the edge. In block **1405**, the routine sets the session of the binding to the variable session. In block **1406**, the routine sets the key of the binding to the key from the session. In block **1407**, the routine sets the entry for the edge in the binding table of the local session to the binding. In block **1408**, the routine invokes the create binding function of the edge of the binding passing the binding so the edge can initialize the binding. If that function returns a status of remove, the routine continues at block **1409**. In block **1409**, the routine sets the binding to be removed and then returns.

FIG. **15** is a flow diagram of the find path routine. The find path routine identifies the shortest path through the binding list. If no such path exists, then the routine extends a path to include the binding list. In decision block **1501**, if the binding is simplex and a path already goes through this binding (returned as an entry), then the routine continues at block **1502**, else the routine continues at block **1503**. In block **1502**, the routine sets the path to the path of the entry and returns. In block **1503**, the routine initializes the pointers element and short entry to null. In block **1504**, the routine sets the path to the path of the passed path entry. If the local path is not null and its status is demux extend, then the routine continues at block **1509**, else the routine continues at block **1505**. In blocks **1505-1508**, the routine loops identifying the shortest path through the bindings in the binding list. The routine loops selecting each path through the binding. The selected path is eligible if it starts at the first binding in the binding list and the path ends at the binding. The routine loops setting the short entry to the shortest eligible path found so far. In block **1505**, the routine sets the variable first binding to the first binding in the binding list of the path address. In block **1506**, the routine selects the next path (entry) in the path list of the binding starting with the first. If a path is selected (indicating that there are more paths in the binding), then the routine continues at block **1507**, else the routine continues at block **1509**. In block **1507**, the routine determines whether the selected path starts at the first binding in the binding list, whether the selected path ends at the last binding in the binding list, and whether the number of path entries in the selected path is less than the number of path entries in the shortest path selected so far. If these conditions are all satisfied, then the routine continues at block **1508**, else the routine loops to block **1506** to select the next path (entry). In block **1508**, the routine sets the shortest path (short entry) to the selected path and loops to block **1506** to select the next path through the binding. In block **1509**, the routine sets the selected path (entry) to the shortest path. In decision block **1510**, if a path has been found, then the routine continues at block **1511**, else the routine continues at block **1512**. In block **1511**, the routine sets the path to the path of the selected path entry and returns. Blocks **1512-1516** are performed when no paths have been found. In block **1512**, the routine sets the path to the path of the passed path entry. If the passed path entry has a path and its status is demux extend, then the routine continues at block **1515**, else the routine continues at block **1513**. In block **1513**, the routine creates a path for the path address. In block **1514**, the routine sets the variable element to null and sets the path entry to the first element in the stack list of the path. In block **1515**, the routine sets the variable element to be address entry of the member of the passed path entry and sets the path entry to the passed path entry. In block **1516**, the routine invokes the extend path routine to extend the path and then returns. The extend path routine creates a path through the bindings of the binding list and sets the path status to the current demux status.

FIG. **16** is a flow diagram of the process of path hopping routine. Path hopping occurs when the path through the binding list is not the same path as that of the passed path entry. In decision block **1601**, if the path of the passed path entry is set, then the routine continues at block **1602**, else the routine continues at block **1609**. In decision block **1602**, if the path of the passed path entry is equal to the local path, then the routine continues at **1612**, else path hopping is occurring and the routine continues at block **1603**. In blocks **1603-1607**, the routine loops positioning pointers at the first path entries of the paths that are not at the same binding. In block **1603**, the routine sets the variable old stack to the stack list of the path of the passed path entry. In block **1604**, the routine sets the variable new stack to the stack list of the local path. In block **1605**, the routine sets the variable old element to the next element in the old stack. In block **1606**, the routine sets the variable element to the next element in the new stack. In decision block **1607**, the routine loops until the path entry that is not in the same binding is located. In decision block **1608**, if the variable old entry is set, then the routine is not at the end of the hopped from path and the routine continues at block **1609**, else routine continues at block **1612**. In block **1609**, the routine sets the variable entry to the previous entry in the hopped-to path. In block **1610**, the routine sets the path of the passed path entry to the local path. In block **1611**, the routine sets the local entry to the first path entry of the stack list of the local path. In block **1612**, the routine inserts an entry into return list and then returns.

Although the conversion system has been described in terms of various embodiments, the invention is not limited to these embodiments. Modification within the spirit of the invention will be apparent to those skilled in the art. For example, a conversion routine may be used for routing a message and may perform no conversion of the message. Also, a reference to a single copy of the message can be passed to each conversion routine or demux key routine. These routines can advance the reference past the header information for the protocol so that the reference is positioned at the next header. After the demux process, the reference can be reset to point to the first header for processing by the conversion routines in sequence. The scope of the invention is defined by the claims that follow.

What is claimed is:

1. A method, comprising:

receiving, at a computing device having a processing circuit, a packet of a message;

determining, by the computing device, a key value for the packet, wherein the key value is determined based on one or more headers in the packet;

using, by the computing device, the key value to determine whether the computing device is currently storing a previously created path for the key value;

in response to determining that no path is currently stored for the key value, the computing device:

identifying, using the key value, one or more routines for processing the packet, including a routine that is used to execute a Transmission Control Protocol (TCP) to convert packets having a TCP format into a different format;

creating a path using the identified one or more routines; and

processing the packet using the created path.

2. The method of claim **1**, wherein the created path stores state information for at least one of the identified one or more routines.

3. The method of claim **1**, wherein the created path stores state information for each of the identified one or more routines.

4. The method of claim **1**, wherein the created path specifies an ordering in which the identified one or more routines are to be performed to process the packet.

5. The method of claim **4**, wherein the ordering specifies that an application layer protocol is to be performed subsequent to the TCP.

6. The method of claim **5**, wherein the application layer protocol is HTTP, and wherein the different format is HTTP.

7. The method of claim **4**, wherein the ordering specifies that a first execution of the TCP is to be followed by execution of an application layer protocol, which is to be followed by a second execution of the TCP.

8. The method of claim **7**, wherein the first execution of the TCP receives information from a network and the second execution of the TCP sends information via the network.

9. The method of claim **4**, wherein the ordering specifies that the TCP is an initial one of the one or more routines.

10. The method of claim **4**, wherein the ordering specifies that the TCP is to be performed after performing an Ethernet protocol.

11. The method of claim **1**, further comprising:

receiving, at the computing device, a subsequent packet of the message;

determining, by the computing device based on the subsequent packet, the key value;

using, by the computing device, the key value to identify the created path for the message; and

processing, by the computing device, the subsequent packet using the path.

12. The method of claim **11**, wherein processing the subsequent packet includes:

queuing the subsequent packet for one or more routines specified in the path; and

performing the one or more routines according to an ordering specified by the path, wherein performing at least one of the routines includes accessing state information stored in the path.

13. The method of claim **11**, wherein packets of the message are all associated with a single TCP session.

14. The method of claim **1**, wherein the key value includes an IP address and one or more port addresses.

15. A method, comprising:

receiving, at a computing device having a processing circuit, a packet of a message;

determining, by the computing device, a key value for the packet, wherein the key value is determined based on one or more headers in the packet;

using, by the computing device, the key value to determine whether the computing device is currently storing a previously created path for the key value;

in response to determining that no path is currently stored for the key value, the computing device:

identifying, using the key value, one or more routines for processing the packet, including a routine that is used to execute a User Datagram Protocol (UDP) to convert packets having a UDP format into a different format;

creating a path using the identified one or more routines; and

processing the packet using the created path.

16. An apparatus, comprising:

a network interface;

a processor circuit;

a memory storing program instructions executable by the processor circuit to:

receiving, via the network interface, a packet of a message;

determine a key value for the packet, wherein the key value is determined based on one or more headers in the packet;

use the key value to determine whether the apparatus is currently storing a path for the key value, wherein one or more routines are specified in the path for processing packets of the message;

in response to determining that no path is currently stored for the key value:

identify, using the key value, one or more routines for processing the packet, including a particular routine that is used to execute a Transmission Control Protocol (TCP) to convert packets having a TCP format into a different format;

create a path using the identified one or more routines;

process the packet using the created path; and

store the path for use in processing subsequent packets in the message; and

in response to determining that a path is currently stored for the key value;

process the packet using the stored path.

17. The apparatus of claim **16**, wherein the apparatus is configured to process the packet by queuing the packet for the one or more routines identified in the path.

18. The apparatus of claim **16**, wherein the different format is an application layer format.

19. The apparatus of claim **16**, wherein the particular routine is executable to utilize state information stored within the path.

20. The apparatus of claim **16**, wherein the path stores state information for at least some of the one or more routines.

* * * * *

EXHIBIT 8

US010033839B2

(12) **United States Patent**
Balassanian

(10) **Patent No.: US 10,033,839 B2**
(45) **Date of Patent: *Jul. 24, 2018**

(54) **METHOD AND SYSTEM FOR DATA DEMULTIPLEXING**

(71) Applicant: **Implicit, LLC**, Seattle, WA (US)

(72) Inventor: **Edward Balassanian**, Seattle, WA (US)

(73) Assignee: **Implicit, LLC**, Seattle, WA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **15/450,790**

(22) Filed: **Mar. 6, 2017**

(65) **Prior Publication Data**

US 2017/0310792 A1 Oct. 26, 2017

Related U.S. Application Data

(63) Continuation of application No. 15/050,027, filed on Feb. 22, 2016, now Pat. No. 9,591,104, which is a

(Continued)

(51) **Int. Cl.**

H04L 12/58	(2006.01)
H04L 29/06	(2006.01)
H04L 29/08	(2006.01)
H04L 29/12	(2006.01)
H04L 12/701	(2013.01)

(52) **U.S. Cl.**

CPC ***H04L 69/08*** (2013.01); ***H04L 29/06*** (2013.01); ***H04L 45/00*** (2013.01); ***H04L 61/2007*** (2013.01); ***H04L 61/6063*** (2013.01); ***H04L 67/02*** (2013.01); ***H04L 69/16*** (2013.01);

(Continued)

(58) **Field of Classification Search**

None

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,298,674 A	3/1994	Yun
5,392,390 A	2/1995	Crozier

(Continued)

FOREIGN PATENT DOCUMENTS

EP	0408132	1/1991
EP	0807347	11/1997

(Continued)

OTHER PUBLICATIONS

Non-Final Office Action in Inter Partes Reexamination Control No. 95/000,659 dated Aug. 16, 2013, 107 pages.

(Continued)

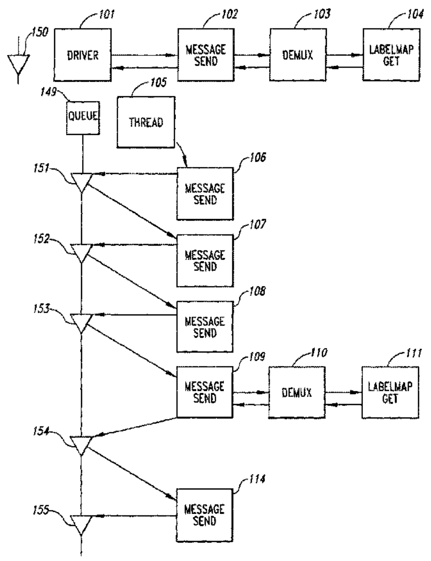
Primary Examiner — Duc T Duong

(74) *Attorney, Agent, or Firm* — Meyertons, Hood, Kivlin, Kowert & Goetzel, P.C.

(57) **ABSTRACT**

The demultiplexing system receives packets of a message, identifies a sequence of message handlers for processing the message, identifies state information associated with the message for each message handler, and invokes the message handlers passing the message and the associated state information. The system identifies the message handlers based on the initial data type of the message and a target data type. The identified message handlers effect the conversion of the data to the target data type through various intermediate data types.

1 Claim, 16 Drawing Sheets



Related U.S. Application Data

continuation of application No. 14/230,952, filed on Mar. 31, 2014, now Pat. No. 9,270,790, which is a continuation of application No. 13/911,324, filed on Jun. 6, 2013, now Pat. No. 8,694,683, which is a continuation of application No. 13/236,090, filed on Sep. 19, 2011, now abandoned, which is a continuation of application No. 10/636,314, filed on Aug. 6, 2003, now Pat. No. 8,055,786, which is a continuation of application No. 09/474,664, filed on Dec. 29, 1999, now Pat. No. 6,629,163.

(52) **U.S. Cl.**
CPC ***H04L 69/18*** (2013.01); ***H04L 69/22*** (2013.01); ***H04L 69/32*** (2013.01)

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,414,833	A		5/1995	Hershey et al.
5,425,029	A		6/1995	Hluchyj et al.
5,568,478	A		10/1996	van Loo, Jr. et al.
5,627,997	A		5/1997	Pearson et al.
5,710,917	A		1/1998	Musa et al.
5,727,159	A		3/1998	Kikinis
5,740,430	A		4/1998	Rosenberg et al.
5,761,651	A		6/1998	Hasebe
5,768,521	A		6/1998	Dedrick
5,826,027	A		10/1998	Pedersen et al.
5,835,726	A		11/1998	Shwed et al.
5,842,040	A		11/1998	Hughes et al.
5,848,233	A		12/1998	Radia et al.
5,848,246	A		12/1998	Gish
5,848,415	A		12/1998	Guck
5,854,899	A		12/1998	Callon et al.
5,870,479	A		2/1999	Feiken et al.
5,896,383	A		4/1999	Wakeland
5,898,830	A		4/1999	Wesinger, Jr. et al.
5,918,013	A		6/1999	Mighdoll et al.
5,983,348	A		11/1999	Ji
5,987,256	A		11/1999	Wu et al.
5,991,299	A		11/1999	Radogna et al.
5,991,806	A		11/1999	McHann, Jr.
6,032,150	A		2/2000	Nguyen
6,035,339	A		3/2000	Agraharam et al.
6,047,002	A		4/2000	Hartmann et al.
6,067,575	A		5/2000	McManis et al.
6,091,725	A		7/2000	Cheriton et al.
6,094,679	A		7/2000	Teng et al.
6,101,189	A		8/2000	Tsuruoka
6,101,320	A		8/2000	Schuetze et al.
6,104,500	A		8/2000	Alam et al.
6,104,704	A		8/2000	Buhler et al.
6,111,893	A		8/2000	Volftsun et al.
6,112,250	A		8/2000	Appelman
6,115,393	A		9/2000	Engel et al.
6,119,165	A		9/2000	Li et al.
6,119,236	A		9/2000	Shipley
6,122,666	A		9/2000	Beurket et al.
6,128,624	A		10/2000	Papierniak et al.
6,130,917	A		10/2000	Monroe
6,141,749	A		10/2000	Coss et al.
6,151,390	A		11/2000	Volftsun et al.
6,157,622	A		12/2000	Tanaka et al.
6,167,441	A		12/2000	Himmel
6,192,419	B1		2/2001	Aditham et al.
6,199,054	B1		3/2001	Khan et al.
6,212,550	B1		4/2001	Segur
6,222,536	B1		4/2001	Kihl et al.
6,226,267	B1		5/2001	Spinney et al.
6,243,667	B1		6/2001	Kerr et al.
6,246,678	B1		6/2001	Erb et al.
6,259,781	B1		7/2001	Crouch et al.
6,275,507	B1		8/2001	Anderson et al.
6,278,532	B1		8/2001	Heimendinger et al.
6,292,827	B1		9/2001	Raz
6,356,529	B1		3/2002	Zarom
6,359,911	B1		3/2002	Movshovich et al.
6,374,305	B1		4/2002	Gupta et al.
6,401,132	B1		6/2002	Bellwood et al.
6,404,775	B1		6/2002	Leslie et al.
6,405,254	B1		6/2002	Hadland
6,426,943	B1		7/2002	Spinney et al.
6,493,348	B1		12/2002	Gelman et al.
6,504,843	B1		1/2003	Cremin et al.
6,519,636	B2		2/2003	Engel et al.
6,560,236	B1		5/2003	Varghese et al.
6,574,610	B1		6/2003	Clayton et al.
6,578,084	B1		6/2003	Moberg et al.
6,598,034	B1		7/2003	Kloth
6,629,163	B1		9/2003	Balassanian
6,650,632	B1		11/2003	Volftsun et al.
6,651,099	B1		11/2003	Dietz et al.
6,678,518	B2		1/2004	Eerola
6,680,922	B1		1/2004	Jorgensen
6,701,432	B1		3/2004	Deng et al.
6,711,166	B1		3/2004	Amir et al.
6,772,413	B2		8/2004	Kuznetsov
6,785,730	B1		8/2004	Taylor
6,865,735	B1		3/2005	Sirer et al.
6,871,179	B1		3/2005	Kist et al.
6,889,181	B2		5/2005	Kerr et al.
6,937,574	B1		8/2005	Delaney et al.
6,957,346	B1		10/2005	Kivinen et al.
6,959,439	B1		10/2005	Boike
7,233,569	B1		6/2007	Swallow
7,233,948	B1		6/2007	Shamoon et al.
7,281,036	B1		10/2007	Lu et al.
7,383,341	B1		6/2008	Saito et al.
7,711,857	B2		5/2010	Balassanian
8,055,786	B2		11/2011	Balassanian
8,694,683	B2		4/2014	Balassanian
2003/0142669	A1		7/2003	Kubota et al.
2004/0015609	A1		1/2004	Brown et al.
2008/0250045	A1		10/2008	Balassanian et al.
2009/0083763	A1		3/2009	Sareen et al.
2009/0265695	A1		10/2009	Karino
2009/0310485	A1	*	12/2009	Averi H04L 45/00 370/232
2015/0032691	A1	*	1/2015	Hall H04L 29/06 707/610

FOREIGN PATENT DOCUMENTS

EP	0817031	1/1998
JP	H10-49354	2/1998
JP	H10-55279	2/1998
JP	H10-74153	3/1998
JP	H10-289215	10/1998
WO	99/35799	7/1999

OTHER PUBLICATIONS

Decision on Petition in Reexamination Control No. 95/000,659 dated Aug. 19, 2013, 3 pages.
Response to Non-Final Office Action in Reexamination Control No. 95/000,659 dated Oct. 2, 2013 including Exhibits A-C, 37 pages.
Decision on Petition in Reexamination Control No. 95/000,660 dated Jul. 30, 2013, 12 pages.
Non-Final Office Action in Inter Partes Reexamination Control No. 95/000,660 dated Aug. 30, 2013, 23 pages.
RFC: 791. Internet Protocol: DARPA Internet Program Protocol Specification, Sep. 1981, prepared for Defense Advanced Research Projects Agency Information Processing Techniques Office by Information Sciences Institute University of Southern California, 52 pages.
2015 WL 2194627, United States District Court, N.D. California, *Implicit L.L.C.*, Plaintiff, v. *F5 Networks, Inc.*, Defendant, Case No. 14-cv-02856-SI, signed May 6, 2015, 14 pages.
Defendants' Invalidity Contentions Pursuant to Local Patent Rules 3-3 and 3-4, United States District Court Eastern District of Texas

(56) **References Cited**

OTHER PUBLICATIONS

Tyler Division, *Implicit, LLC* v. *Trend Micro, Inc., Ericsson Inc., Huawei Technologies Usa, Inc., NEC corporation of America, Nokia Solutions and Networks US LLC*; Sep. 2, 2016, 53 pages.
Exhibits A-1-A16 Invalidity of U.S. Pat. No. 8,694,683, Defendants' Invalidity Contentions Pursuant to Local Patent Rules 3-3 and 3-4, United States District Court Eastern District of Texas Tyler Division, Sep. 2, 2016, 425 pages.
Exhibits B-1-B13 Invalidity of U.S. Pat. No. 9,270,790, Defendants' Invalidity Contentions Pursuant to Local Patent Rules 3-3 and 3-4, United States District Court Eastern District of Texas Tyler Division, Sep. 2, 2016, 369 pages.
Exhibits C-1-C21 Invalidity of U.S. Pat. No. 8,856,779, Defendants' Invalidity Contentions Pursuant to Local Patent Rules 3-3 and 3-4, United States District Court Eastern District of Texas Tyler Division, Sep. 2, 2016, 646 pages.
Exhibits D-1-D21 Invalidity of U.S. Pat. No. 9,325,740, Defendants' Invalidity Contentions Pursuant to Local Patent Rules 3-3 and 3-4, United States District Court Eastern District of Texas Tyler Division, Sep. 2, 2016, 419 pages.
Exhibits E-1-E20 Invalidity of U.S. Pat. No. 6,324,685, Defendants' Invalidity Contentions Pursuant to Local Patent Rules 3-3 and 3-4, United States District Court Eastern District of Texas Tyler Division, Sep. 2, 2016, 416 pages.
Michael Baentsch, et al., "WebMake: Integrating distributed software development in a structure-enhanced Web," Computer Networks and ISDN Systems 27 (1995), pp. 789-800.
Dan Decasper, et al., "A Scalable, High Performance Active Network Node," Apr. 1998, 21 pages.
John J. Hartman, et al., "Joust: A Platform for Liquid Software," Computer, IEEE, 1999, pp. 50-56.
David Mosberger, et al., "Making Paths Explicit in the Scout Operating System," Proceedings of the USENIX 2nd Symposium on Operating Systems Design and Implementation, Oct. 1996, 16 pages.
Oliver Spatscheck, et al., "Escort: A Path-Based OS Security Architecture," TR 97-17, Nov. 26, 1997, 17 pages.
Dan Decasper, et al., "DAN: Distrubuted Code Caching for Active Networks," IEEE, 1998, pp. 609-616.
Alexander, D. et al., "The SwitchWare Active Network Architecture", Jun. 6, 1998, IEEE.
Antoniazzi, S. et al., "An Open Software Architecture for Multimedia Consumer Terminals", Central Research Labs, Italy; Alcatel Sel Research Centre, Germany, ECMAST 1997.
Arbanowski, Stefan, "Generic Description of Telecommunication Services and Dynamic Resource Selection in Intelligent Communication Environments", Thesis, Technische Universitat Berlin, Oct. 9, 1996 (3 documents).
Arbanowski, S., et al., Service Personalization for Unified Messaging Systems, Jul. 6-8, 1999, The Fourth IEEE Symposium on Computers and Communications, ISCC '99, Red Sea, Egypt.
Atkinson, R., "Security Architecture for the Internet Protocol", Aug. 1995, Naval Research Laboratory.
Atkinson, R., "IP Authentication Header", Aug. 1995, Naval Research Laboratory.
Atkinson, R., "IP Encapsulating Security Payload (ESP)", Aug. 1995, Naval Research Laboratory.
Back, G., et al., Java Operating Systems: Design and Implementation, Aug. 1998, Technical Report UUCS-98-015, University of Utah.
Baker, Dr. Sean, "CORBA Implementation Issues", 1994, IONA Technologies, O'Reilly Institute Dublin, Ireland.
Barrett, R., et al., "Intermediaries: New Places for Producing and Manipulating Web Content", 1998, IBM Almaden Research Center, Elsevier Science.
Bellare, M., et al., "A Concrete Security Treatment of Symmetric Encryption: Analysis of the DES Modes of Operation", Aug. 15, 1997, Dept. of Computer Science and Engineering, University of California, San Diego.
Bellare, M., et al., "A Concrete Security Treatment of Symmetric Encryption: Analysis of the DES Modes of Operation", Aug. 15, 1997, IEEE.
Bellare, M., et al., "XOR MACs: New Methods for Message Authentication Using Finite Pseudorandom Functions", 1995, CRYPTO '95, LNCS 963, pp. 15-28, Springer-Verlag Berlin Heidelberg.
Bellissard, L., et al., "Dynamic Reconfiguration of Agent-Based Applications", Third European Research Seminar on Advances in Distributed Systems, (ERSADS '99) Madeira Island.
Bolding, Darren, "Network Security, Filters and Firewalls", 1995, www.acm.org/crossroads/xrds2-1/security.html.
Booch, G., et al., "Software Engineering with ADA", 1994, Third Edition, The Benjamin/Cummings Publishing Company, Inc (2 documents).
Breugst, et al., "Mobile Agents—Enabling Technology for Active Intelligent Network Implementation", May/Jun. 1998, IEEE Network.
"C Library Functions", AUTH(3) Sep. 17, 1993, Solbourne Computer, Inc.
Chapman, D., et al., "Building Internet Firewalls", Sep. 1995, O'Reilly & Associates, Inc.
CheckPoint FireWall-1 Technical White Paper, Jul. 18, 1994, CheckPoint Software Technologies, Ltd.
CheckPoint FireWall-1 White Paper, Sep. 1995, Version 2.0, CheckPoint Software Technologies, Ltd.
Command Line Interface Guide P/N 093-0011-000 Rev C Version 2.5, 2000-2001, NetScreen Technologies, Inc.
Coulson, G. et al., "A CORBA Compliant Real-Time Multimedia Platform for Broadband Networks", Lecture Notes in Computer Science, 1996, Trends in Distributed Systems CORBA and Beyond.
Cox, Brad, "SuperDistribution, Objects As Property on the Electronic Frontier", 1996, Addison-Wesley Publishing Company.
Cranes, et al., "A Configurable Protocol Architecture for CORBA Environments", Autonomous Decentralized Systems 1997 Proceedings ISADS, Third International Symposium Apr. 9-11, 1997.
Curran, K., et at, "CORBA Lacks Venom", University of Ulster, Northern Ireland, UK 2000.
Dannert, Andreas, "Call Logic Service for a Personal Communication Supporting System", Thesis, Jan. 20, 1998, Technische Universitat Berlin.
DARPA Internet Program Protocol Specification, "Transmission Control Protocol", Sep. 1981, Information Sciences Institute, California.
DARPA Internet Program Protocol Specification, "Internet Protocol", Sep. 1981, Information Sciences Institute, California.
Decasper, D., et al., "Crossbow: A Toolkit for Integrated Services over Cell Switched IPv6", 1997, Computer Engineering and Networks Laboratory, ETH Zurich, Switzerland.
Decasper, D., et al., "Router Plugins a Software Architecture for Next Generation Routers", 1998, Proceedings of ACM SIGCONM '98.
Deering, S., et al., Internet Protocol, Version 6 (IPv6) Specification, Dec. 1998, Nokia, The Internet Society.
Deering, S., et al., Internet Protocol, Version 6 (IPv6) Specification, Dec. 1995, Network Working Group, RFC 1883.
Dutton, et al, "Asynchronous Transfer Mode Technical Overview (ATM)", Second Edition; IBM, Oct. 1995, 2nd Edition, Prentice Hall PTR, USA.
Eckardt, T., et al., "Application of X.500 and X.700 Standards for Supporting Personal Communications in Distributed Computing Environments", 1995, IEEE.
Eckardt, T., et al., "Personal Communications Support based on TMN and TINA Concepts", 1996, IEEE Intelligent Network Workshop (IN '96), Apr. 21-24, Melbourne, Australia.
Eckardt, T., et al., "Beyond IN and UPT—A Personal Communications Support System Based on TMN Concepts", Sep. 1997, IEEE Journal on Selected Areas in Communications, vol. 15, No. 7.
Egevang, K., et al., "The IP Network Address Translator (NAT)", May 1994, Network Working Group, RFC 1631.

(56) **References Cited**

OTHER PUBLICATIONS

Estrin, D., et al., "Visa Protocols for Controlling Inter-Organizational Datagram Flow", Dec. 1998, Computer Science Department, University of Southern California and Digital Equipment Corporation.
Faupel, M., "Java Distribution and Deployment", Oct. 9, 1997, APM Ltd., United Kingdom.
Felber, P., "The CORBA Object Group Service: A Service Approach to Object Groups in CORBA", Thesis, 1998, Ecole Polytechnique Federale de Lausanne, Switzerland.
Fish, R., et al., "DRoPS: Kernel Support for Runtime Adaptable Protocols", Aug. 25-27, 1998, IEEE 24th Euromicro Conference, Sweden.
Fiuczynski, M., et al., "An Extensible Protocol Architecture for Application-Specific Networking", 1996, Department of Computer Science and Engineering, University of Washington.
Franz, Stefan, "Job and Stream Control in Heterogeneous Hardware and Software Architectures", Apr. 1998, Technische Universitat, Berlin (2 documents).
Fraser, T., "DTE Firewalls: Phase Two Measurement and Evaluation Report", Jul. 22, 1997, Trusted Information Systems, USA.
Gazis, V., et al., "A Survey of Dynamically Adaptable Protocol Stacks", first Quarter 2010, IEEE Communications Surveys & Tutorials, vol. 12, No. 1, 1st Quarter.
Gokhale, A., et al., "Evaluating the Performance of Demultiplexing Strategies for Real-Time CORBA", Nov. 1997, GLOBECOM.
Gokhale, A., et al., "Measuring and Optimizing CORBA Latency and Scalability Over High-Speed Networks", Apr. 1998, IEEE Transaction on Computers, vol. 47, No. 4; Proceedings of the International Conference on Distributed Computing Systems (ICDCS '97) May 27-30, 1997.
Gokhale, A., et al., "Operating System Support for High-Performance, Real-Time CORBA", 1996.
Gokhale, A., et al., "Principles for Optimizing CORBA Internet Inter-ORB Protocol Performance", Jan. 9, 1998, Proceedings of the HICSS Conference, Hawaii.
Gong, Li, "Java Security: Present and Near Future", May/Jun. 1997, IEEE Micro.
Gong, Li, "New Security Architectural Directions for Java (Extended Abstract)", Dec. 19, 1996, IEEE.
Gong, Li, "Secure Java Class Loading", Nov./Dec. 1998, IEEE Internet.
Goos, G., et al., "Lecture Notes in Computer Science: Mobile Agents and Security", 1998, Springer-Verlag Berlin Heidelberg.
Goralski, W., "Introduction to ATM Networking", 1995, McGraw-Hill Series on Computer Communications, USA.
Hamzeh, K., et al., "Layer Two Tunneling Protocol L2TP", Jan. 1998, PPP Working Group, Internet Draft.
Harrison, T., et al., "The Design and Performance of a Real-Time CORBA Event Service", Aug. 8, 1997,Proceedings of the OOPSLA '97 Conference, Atlanta, Georgia in Oct. 1997.
Huitema, Christian, "IPv6 The New Internet Protocol", 1997 Prentice Hall, Second Edition.
Hutchins, J., et al., "Enhanced Internet Firewall Design Using Stateful Filters Final Report", Aug. 1997, Sandia Report; Sandia National Laboratories.
IBM, Local Area Network Concepts and Products: Routers and Gateways, May 1996.
Juniper Networks Press Release, Juniper Networks Announces Junos, First Routing Operating System for High-Growth Internet Backbone Networks, Jul. 1, 1998, Juniper Networks.
Juniper Networks Press Release, Juniper Networks Ships the Industry's First Internet Backbone Router Delivering Unrivaled Scalability, Control and Performance, Sep. 16, 1998, Juniper Networks.
Karn, P., et al., "The ESP DES-CBC Transform", Aug. 1995, Network Working Group, RFC 1829.
Kelsey, J. et al., "Authenticating Outputs of Computer Software Using a Cryptographic Coprocessor", Sep. 1996, CARDIS.
Krieger, D., et al., "The Emergence of Distributed Component Platforms", Mar. 1998, IEEE.
Krupczak, B., et al., "Implementing Communication Protocols in Java", Oct. 1998, IEEE Communications Magazine.
Krupczak, B., et al., "Implementing Protocols in Java: The Price of Portability", 1998, IEEE.
Lawson, Stephen, "Cisco NetFlow Switching Speeds Traffic Routing", Jul. 7, 1997, Infoworld.
Li, S., et al., "Active Gateway: A Facility for Video Conferencing Traffic Control", Feb. 1, 1997, Purdue University; Purdue e-Pubs; Computer Science Technical Reports.
Magedanz, T., et al., "Intelligent Agents: An Emerging Technology for Next Generation Telecommunications?", 1996, IEEE.
Mills, H., et al., "Principles of Information Systems Analysis and Design", 1986, Academic Press, Inc (2 documents).
Mosberger, David, "*Scout: A Path-Based Operating System*", Doctoral Dissertation Submitted to the University of Arizona, 1997 (3 documents).
Muhugusa, M., et al., "ComScript : An Environment for the Implementation of Protocol Stacks and their Dynamic Reconfiguration", Dec. 1994.
Nelson, M., et al., The Data Compression Book, 2nd Edition, 1996, M&T Books, a division of MIS Press, Inc.
NetRanger User's Guide, 1996, WheelGroup Corporation.
NetScreen Command Line Reference Guide, 2000, P/N 093-0000-001 Rev A, NetScreen Technologies, Inc., USA.
NetScreen Command Line Reference Guide, 2000, P/N 093-0000-001 NetScreen Technologies, Inc., USA.
NetScreen Concepts and Examples ScreenOS Reference Guide, 1998-2001, Version 2.5 P/N 093-0039-000 Rev. A, NetScreen Technologies, Inc.
NetScreen Products Webpage, wysiwyg://body_bottom.3/http://www . . . een.com/products/products.html 1998-1999, NetScreen Technologies, Inc.
NetScreen WebUI, Reference Guide, Version 2.5.0 P/N 093-0040-000 Rev. A, 2000-2001, NetScreen Technologies, Inc.
NetStalker Installation and User's Guide, 1996, Version 1.0.2, Haystack Labs, Inc.
Niculescu, Dragos, "Survey of Active Network Research", Jul. 14, 1999, Rutgers University.
Nortel Northern Telecom, "ISDN Primary Rate User-Network Interface Specification", Aug. 1998.
Nygren, Erik, "The Design and Implementation of a High-Performance Active Network Node", Thesis, Feb. 1998, MIT.
Osboume, E., "Morningstar Technologies SecureConnect Dynamic Firewall Filter User's Guide", Jun. 14, 1995, V. 1.4, Morning Star Technologies, Inc.
Padovano, Michael, "Networking Applications on UNIX System V Release 4," 1993 Prentice Hall, USA (2 documents).
Pfeifer, T., "Automatic Conversion of Communication Media", 2000, GMD Research Series, Germany.
Pfeifer, T., "Automatic Conversion of Communication Media", Thesis, 1999, Technischen Universitat Berlin, Berlin.
Pfeifer, T., et al., "Applying Quality-of-Service Parametrization for Medium-to-Medium Conversion", Aug. 25-28, 1996, 8th IEEE Workshop on Local and Metropolitan Area Networks, Potsdam, Germany.
Pfeifer, T., "Micronet Machines—New Architectural Approaches for Multimedia End-Systems", 1993 Technical University of Berlin.
Pfeifer, T., "On the Convergence of Distributed Computing and Telecommunications in the Field of Personal Communications", 1995, KiVS, Berlin.
Pfeifer, T., "Speech Synthesis in the Intelligent Personal Communication Support System (IPCSS)", Nov. 2-3, 1995, 2nd 'Speak!' Workshop on Speech Generation in Multimodal Information Systems and Practical Applications.
Pfeifer, T., et al., "Generic Conversion of Communication Media for Supporting Personal Mobility", Nov. 25-27, 1996, Proc. of the Third COST 237 Workshop: Multimedia Telecommunications and Applications.
Pfeifer, T., et al., "Intelligent Handling of Communication Media", Oct. 29-31, 1997, 6th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS) Tunis.
Pfeifer, T., et al., "Resource Selection in Heterogeneous Communication Environments using the Teleservice Descriptor", Dec.

(56) **References Cited**

OTHER PUBLICATIONS

15-19, 1997, Proceedings from the 4th COST 237 Workshop: From Multimedia Services to Network Services, Lisboa.
Pfeifer, T., et al., Mobile Guide—Location-Aware Applications from the Lab to the Market, 1998, IDMS '98, LNCS 1483, pp. 15-28.
Pfeifer, T., et al., "The Active Store providing Quality Enhanced Unified Messaging", Oct. 20-22, 1998, 5th Conference on computer Communications, AFRICOM-CCDC '98, Tunis.
Pfeifer, T.,, et al., "*A Modular Location-Aware Service and Application Platform*", 1999, Technical University of Berlin.
Plagemann, T., et at, "*Evaluating Crucial Performance Issues of Protocol Configuration in DaCaPo*", 1994, University of Oslo.
Psounis, Konstantinos, "*Active Networks: Applications, Security, Safety, and Architectures*", First Quarter 1999, IEEE Communications Surveys.
Rabiner, Lawrence, "*Applications of Speech Recognition in the Area of Telecommunications*", 1997, IEEE.
Raman, Suchitra, et al, "*A Model, Analysis, and Protocol Framework for Soft State-based Communications*", Department of EECS, University of California, Berkeley.
Rogaway, Phillip, "*Bucket Hashing and its Application to Fast Message Authentication*", Oct. 13, 1997, Department of Computer Science, University of California.
Schreier, B., et al., "*Remote Auditing of Software Outputs Using a Trusted CoProcessor*", 1997, Elsevier Paper Reprint 1999.
Tennenhouse, D., et al., "*From Internet to ActiveNet*", Laboratory of Computer Science, MIT, 1996.
Tudor, P., "*Tutorial MPEG-2 Video Compression*", Dec. 1995, Electronics & Communication Engineering Journal.
US Copyright Webpage of Copyright Title, "*IPv6: the New Internet Protocol*", by Christian Huitema, 1998 Prentice Hall.
Van der Meer, et al., "*An Approach for a 4th Generation Messaging System*", Mar. 21-23, 1999, The Fourth International Symposium on Autonomous Decentralized Systems ISADS '99, Tokyo.
Van der Meer, Sven, "*Dynamic Configuration Management of the Equipment in Distributed Communication Environments*", Thesis, Oct. 6, 1996, Berlin (3 documents).
Van Renesse, R. et al., "*Building Adaptive Systems Using Ensemble*", Cornell University Jul. 1997.
Venkatesan, R., et al., "*Threat-Adaptive Security Policy*", 1997, IEEE.
Wetherall, D., et al., "*The Active IP Option*", Sep. 1996, Proceedings of the 7th ACM SIGOPS European Workshop, Connemara, Ireland.
Welch, Terry, "*A Technique for High-Performance Data Compression*", 1984, Sperry Research Center, IEEE.
Zeletin, R. et al., "*Applying Location Aware Computing for Electronic Commerce: Mobile Guide*", Oct. 20-22, 1998, 5th Conference on Computer Communications, AFRICOM-CCDC '98, Tunis.
Zell, Markus, "*Selection of Converter Chains by Means of Quality of Service Analysis*", Thesis, Feb. 12, 1998, Technische Universitat Berlin.
Implicit Networks, Inc. v. *Advanced Micro Devices, Inc. et al.*; C08-0184 JLR; USDC for the Western District of Washington, Seattle Division.
Feb. 4, 2008 Plaintiff's Original Complaint.
Aug. 26, 2008 Defendant Nvidia Corporation's Answer to Complaint.
Aug. 26, 2008 Defendant Sun Microsystems, Inc.'s Answer to Complaint.
Aug. 27, 2008 Defendant Advanced Micro Devices, Inc.'s Answer to Complaint for Patent Infringement.
Aug. 27, 2008 RealNetworks, Inc.'s Answer to Implicit Networks, Inc.'s Original Complaint for Patent Infringement, Affirmative Defenses, and Counterclaims.
Aug. 27, 2008 Intel Corp.'s Answer, Defenses and Counterclaims.
Aug. 27, 2008 Defendant Rmi Corporation's Answer to Plaintiff's Original Complaint.
Sep. 15, 2008 Plaintiff's Reply to NVIDIA Corporation's Counterclaims.
Sep. 15, 2008 Plaintiff's Reply to Sun Microsystems Inc.'s Counterclaims.
Sep. 16, 2008 Plaintiff's Reply to RealNetworks, Inc.'s Counterclaims.
Sep. 16, 2008 Plaintiff's Reply to Intel Corp.'s Counterclaims.
Dec. 10, 2008 Order granting Stipulated Motion for Dismissal with Prejudice re Nvidia Corporation, Inc.
Dec. 16, 2008 Defendants AMD, RealNetworks, RMI, and Sun's Motion to Stay Pending the Patent and Trademark Office's Reexamination of the '163 Patent.
Dec. 29, 2008 Order granting Stipulated Motion for Dismissal without Prejudice of Claims re Sun Microsystems, Inc.
Jan. 5, 2009 Plaintiff's Opposition to Defendants AMD, RealNetworks, RMI, and Sun's Motion to Stay Pending Reexamination and Exhibit A.
Jan. 9, 2009 Reply of Defendants AMD, RealNetworks, RMI, and Sun's Motion to Stay Pending the Patent and Trademark Office's Reexamination of the '163 Patent.
Feb. 9, 2009 Order Granting Stay Pending the United States Patent and Trademark Office's Reexamination of U.S. Pat. No. 6,629,163.
Feb. 17, 2009 Order Granting Stipulated Motion for Dismissal of Advanced Micro Devices, Inc. with Prejudice.
May 14, 2009 Order Granting Stipulated Motion for Dismissal of RMI Corporation with Prejudice.
Oct. 13, 2009 Order Granting Stipulated Motion for Dismissal of Claims Against and Counterclaims by Intel Corporation.
Oct. 30, 2009 Executed Order for Stipulated Motion for Dismissal of Claims Against and Counterclaims by RealNetworks, Inc.
Implicit Networks, Inc. v. *Microsoft Corp.*, C09-5628 HLR; USDC for the Northern District of California, San Francisco Division.
Nov. 30, 2009 Plaintiff's Original Complaint, *Implicit* v *Microsoft*, Case No. Sep. 5628.
Jan. 22, 2010 Order Dismissing Case, *Implicit* v *Microsoft*, Case No. Sep. 5628.
Implicit Networks, Inc. v. *Cisco Systems, Inc.*, C10-3606 HRL; USDC for the Northern District of California, San Francisco Division.
Aug. 16, 2010 Plaintiff's Original Complaint, *Implicit* v *Cisco*, Case No. 10-3606.
Nov. 22, 2010 Defendant Cisco Systems, Inc.'s Answer and Counterclaims, *Implicit* v *Cisco*, Case No. 10-3606.
Dec. 13, 2010 Plaintiff, Implicit Networks, Inc.'s, Answer to Counterclaims, *Implicit* v *Cisco*, Case No. 10-3606.
Oct. 4, 2011 Order of Dismissal with Prejudice, *Implicit* v *Cisco*, Case No. 10-3606.
Implicit Networks, Inc. v. *Citrix Systems, Inc.*, C10-3766 JL; USDC for the Northern District of California, San Francisco Division.
Aug. 24, 2010 Plaintiff's Original Complaint, *Implicit* v *Citrix*, Case No. 10-3766.
Dec. 1, 2010 Plaintiff's First Amended Complaint, *Implicit* v *Citrix*, Case No. 10-3766.
Jan. 14, 2011 Defendant Citrix Systems, Inc.'s Answer, Defenses and Counter-complaint for Declaratory Judgment, *Implicit* v *Citrix*, Case No. 10-3766.
Feb. 18, 2011 Plaintiff, Implicit Networks, Inc.'s, Answer to Defendants Counterclaims, *Implicit* v *Citrix*, Case No. 10-3766.
May 2, 2011 Order of Dismissal, *Implicit* v *Citrix*, Case No. 10-3766.
Implicit Networks, Inc. v.*F5 Networks, Inc.*, C10-3365 JCS; USDC for the Northern District of California, San Francisco Division.
Jul. 30, 2010 Plaintiff's Original Complaint, *Implicit* v *F5*, Case No. 10-3365.
Oct. 13, 2010 Defendants' Answer and Counter-Complaint, *Implicit* v *F5*, Case No. 10-3365.
Nov. 3, 2010 Plaintiff's Answer to Counter-Complaint, *Implicit* v *F5*, Case No. 10-3365.
Dec. 10, 2010 Plaintiff's First Amended Complaint, *Implicit* v *F5*, Case No. 10-3365.
Jan. 14, 2011 Defendants' Answer to 1st Amended Complaint and Counterclaim, *Implicit* v *F5*, Case No. 10-3365.
Feb. 18, 2011 Plaintiff's Answer to F5' s Amended Counter-Complaint, *Implicit* v *F5*, Case No. 10-3365.

(56) **References Cited**

OTHER PUBLICATIONS

Apr. 18, 2011 Defendants' Amended Answer to 1st Amended Complaint and Counter-Complaint, *Implicit* v *F5*, Case No. 10-3365.
May 5, 2011 Plaintiff's Answer to F5' s Amended Counter-Complaint, *Implicit* v *F5*, Case No. 10-3365.
Jul. 22, 2011 F5 Networks, Inc.'s Invalidity Contentions, *Implicit* v *F5*, Case No. 10-3365.
Jul. 22, 2011 F5 Networks, Inc.'s Invalidity Contentions, Exhibit A, *Implicit* v *F5*, Case No. 10-3365 (31 documents).
Jul. 22, 2011 F5 Networks, Inc.'s Invalidity Contentions, Exhibit B, *Implicit* v *F5*, Case No. 10-3365.
Oct. 18, 2011 Joint Claim Construction & Pre-Hearing Statement (PR 4-3), *Implicit* v *F5*, Case No. 10-3365.
Oct. 18, 2011 Joint Claim Construction & Pre-Hearing Statement (PR 4-3) Exhibit A, *Implicit* v *F5*, Case No. 10-3365 (2 documents).
Nov. 28, 2011 Plaintiff's Opening Claim Construction Brief, *Implicit* v *F5*, Case No. 10-3365.
Nov. 29, 2011 Amended Joint Claim Construction & Pre-Hearing Statement, *Implicit* v *F5*, Case No. 10-3365.
Nov. 29, 2011 Amended Joint Claim Construction & Pre-Hearing Statement, Exhibit A, *Implicit* v *F5*, Case No. 10-3365.
Dec. 12, 2011 Defendants' Claim Construction Brief, *Implicit* v *F5*, Case No. 10-3365.
Dec. 19, 2011 Plaintiff's Reply to Defendants' (F5, HP, Juniper) Responsive Claim Construction Brief (4-5), *Implicit* v *F5*, Case No. 10-3365.
Jan. 27, 2012 Transcript of Proceeding Held on Jan. 17, 2012; *Implicit* v *F5*, Case No. 10-3365.
Jan. 27, 2012 Transcript of Proceeding Held on Jan. 18, 2012; *Implicit* v *F5*, Case No. 10-3365.
Jan. 27, 2012 Transcript of Proceeding Held on Jan. 19, 2012; *Implicit* v *F5*, Case No. 10-3365.
Feb. 29, 2012 Claim Construction Order.
Aug. 15, 2012 Storer Invalidity Report.
Sep. 10, 2012 Implicit's Expert Report of Scott M. Nettles.
Mar. 13, 2013 Order Granting Defendants' Motion for Summary Judgment.
Apr. 9, 2013 Notice of Appeal to the Federal Circuit.
Implicit Networks, Inc. v. *Hewlett-Packard Company*, C10-3746 JCS: USDC for the Northern District of California, San Francisco Division.
Aug. 23, 2010 Plaintiff's Original Complaint, *Implicit* v *HP*, Case No. 10-3746.
Nov. 23, 2010 Plaintiff's First Amended Complaint, *Implicit* v *HP*, Case No. 10-3746.
Jan. 14, 2011 Defendant HP's Answer and Counterclaims, *Implicit* v *HP*, Case No. 10-3746.
Feb. 18, 2011 Implicit Networks, Inc.'s Answer to HP Counterclaims, *Implicit* v *HP*, Case No. 10-3746.
May 10, 2011 Plaintiff's Amended Disclosure of Asserted Claims and Infringement Contentions, Case No. 10-3746.
Jun. 30, 2011 Defendant HP Company's Invalidity Contentions, *Implicit* v *HP*, Case No. 10-3746.
Jun. 30, 2011 Defendant HP Company's Invalidity Contentions, A1-14, *Implicit* v *HP*, Case No. 10-3746.
Jun. 30, 2011 Defendant HP Company's Invalidity Contentions, B1-21, *Implicit* v *HP*, Case No. 10-3746.
Implicit Networks, Inc. v. *Juniper Networks*, C10-4234 EDL: USDC for the Northern District of California, San Francisco Division.
Sep. 20, 2010 Plaintiff's Original Complaint, *Implicit* v *Juniper*, Case No. 10-4234.
Nov. 12, 2010 Juniper Network's Motion to Dismiss for Failure to State a Claim Under Rule 12(B)(6): Memorandum of Points and Authorities; *Implicit* v *Juniper*, Case No. 10-4234.
Nov. 12, 2010 Juniper Network's Request for Judicial Notice in Support of its Motion to Dismiss for Failure to State a Claim Under Rule 12(B)(6): Memorandum of Points and Authoritites; *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 1, 2010 First Amended Complaint; *Implicit* v *Juniper*, Case No. 10-4234.
Jan. 18, 2011 Juniper Networks, Inc.'s Answer and Affirmative Defenses to 1st Amended Complaint, *Implicit* v *Juniper*, Case No. 10-4234.
Feb. 18, 2011 Plaintiff's Answer to Defendant's Counterclaims, *Implicit* v *Juniper*, Case No. 10-4234.
May 23, 2011 Plaintiff's Disclosure of Asserted Claims and Infringement Contentions, *Implicit* v *Juniper*, Case No. 10-4234.
Nov. 15, 2011 Plaintiff's Amended Disclosure of Asserted Claim and Infringement Contentions, *Implicit* v *Juniper*, Case No. 10-4234.
Nov. 28, 2011 Spencer Hosie Declaration in Support of Plaintiffs Opening Claim Construction Brief), *Implicit* v *Juniper*, Case No. 10-4234.
Nov. 28, 2011 Spencer Hosie Declaration in Support of Plaintiffs Opening Claim Construction Brief Exhibit E, *Implicit* v *Juniper*, Case No. 10-4234.
Nov. 28, 2011 Spencer Hosie Declaration in Support of Plaintiffs Opening Claim Construction Brief Exhibit J, *Implicit* v *Juniper*, Case No. 10-4234.
Nov. 28, 2011 Spencer Hosie Declaration in Support of Plaintiffs Opening Claim Construction Brief Exhibit K, *Implicit* v *Juniper*, Case No. 10-4234.
Nov. 28, 2011 Spencer Hosie Declaration in Support of Plaintiffs Opening Claim Construction Brief Exhibits M-O, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit B, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit F, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit N, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit P, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit Q, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit S., *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit T-1, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit T-2, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit T-3, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit T-4, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit U, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit V, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit W, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit X, *Implicit* v *Juniper*, Case No. 10-4234.

(56) **References Cited**

OTHER PUBLICATIONS

Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit Y-1, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit Y-2, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit Y-3, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit Y-4, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit Z, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 19, 2011 Spencer Hosie Declaration in Support of Plaintiff's Reply Claim Construction Brief, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 19, 2011 Spencer Hosie Declaration in Support of Plaintiff's Reply Claim Construction Brief, Exhibit P, *Implicit* v *Juniper*, Case No. 10-4234.
Jan. 10, 2012 Plaintiff's Jan. 10, 2012 Amended Disclosure of Asserted Claims and Infringement Contentions, *Implicit* v *Juniper*, Case No. 10-4234.
Feb. 10, 2012 Juniper Networks, Inc.'s Supplemental Invalidity Contentions, *Implicit* v *Juniper*, Case No. 10-4234.
Feb. 10, 2012 Juniper Networks, Inc.'s Supplemental Invalidity Contentions, Exhibit A1, *Implicit* v *Juniper*, Case No. 10-4234.
Feb. 10, 2012 Juniper Networks, Inc.'s Supplemental Invalidity Contentions, Exhibit A2, *Implicit* v *Juniper*, Case No. 10-4234.
Feb. 10, 2012 Juniper Networks, Inc.'s Supplemental Invalidity Contentions, Exhibit A3, *Implicit* v *Juniper*, Case No. 10-4234.
Feb. 10, 2012 Juniper Networks, Inc.'s Supplemental Invalidity Contentions, Exhibit A4, *Implicit* v *Juniper*, Case No. 10-4234.
Feb. 10, 2012 Juniper Networks, Inc.'s Supplemental Invalidity Contentions, Exhibit B1, *Implicit* v *Juniper*, Case No. 10-4234.
Feb. 29, 2012 Plaintiff's Feb. 29, 2012 Amended Disclosure of Asserted Claims and Infringement Contentions, *Implicit* v *Juniper*, Case No. 10-4234.
Apr. 6, 2012 Plaintiff's Apr. 6, 2012 Amended Disclosure of Asserted Claims and Infringement Contentions, *Implicit* v *Juniper*, Case No. 10-4234.
Apr. 9, 2012 Plaintiff's Apr. 9, 2012 Amended Disclosure of Asserted Claims and Infringement Contentions, *Implicit* v *Juniper*, Case No. 10-4234.
Sep. 11, 2012 Implicit's Expert Report of Scott Nettles.
Nov. 9, 2012 Juniper's Notice of Motion and Memorandum of Law ISO Motion for Summary Judgment or, in the alternative, for Partial Summary Judgment, on the Issue of Invalidity.
Nov. 9, 2012 Exhibit 2 to Declaration in support of Juniper's Motion for Summary Judgment—Calvert Expert Report.
Nov. 9, 2012 Exhibit 3 to Declaration in support of Juniper's Motion for Summary Judgment—Calvert Supplemental Expert Report.
Nov. 26, 2012 Implicit Opposition to Juniper's and F5 Motion on Invalidity.
Nov. 26, 2012 Exhibit A to Hosie Declaration—Aug. 27, 2012 Excerpts from David Blaine deposition.
Nov. 26, 2012 Exhibit B to Hosie Declaration—Oct. 25, 2012 Excerpts from Kenneth Calvert Deposition.
Nov. 26, 2012 Exhibit C to Hosie Declaration—Aug. 15, 2012 Excerpts from Kenneth Calvert Expert Report.
Nov. 26, 2012 Exhibit D to Hosie Declaration—U.S. Pat. No. 6,651,099 to Dietz et al.
Nov. 26, 2012 Exhibit E to Hosie Declaration—Understanding Packet-Based and Flow-Based Forwarding.
Nov. 26, 2012 Exhibit F to Hosie Declaration—Wikipedia on Soft State.
Nov. 26, 2012 Exhibit G to Hosie Declaration—Sprint Notes.
Nov. 26, 2012 Exhibit H to Hosie Declaration—Implicit's Supplemental Response to Juniper's 2nd Set of Interrogatories.
Nov. 26, 2012 Exhibit Ito Hosie Declaration—U.S. Pat. No. 7,650,634 (Zuk).
Other Implicit Networks, Inc. Prosecution Matters.
U.S. Appl. No. 11/933,022, filed Oct. 31, 2007.
U.S. Appl. No. 11/933,022 Preliminary Amendment dated Feb. 19, 2008.
U.S. Appl. No. 11/933,022 Office Action dated Jun. 24, 2009.
U.S. Appl. No. 11/933,022 Amendment dated Sep. 24, 2009.
U.S. Appl. No. 11/933,022 Office Action dated Dec. 11, 2009.
U.S. Appl. No. 11/933,022 Amendment and Response dated Jan. 29, 2010.
U.S. Appl. No. 11/933,022 Notice of Allowance dated Mar. 2, 2010.
U.S. Appl. No. 11/933,022 Issue Notification dated May 4, 2010.
U.S. Appl. No. 10/636,314, filed Aug. 6, 2003.
U.S. Appl. No. 10/636,314 Office Action dated Apr. 7, 2008.
U.S. Appl. No. 10/636,314 Response to Restriction Requirement dated Aug. 5, 2008.
U.S. Appl. No. 10/636,314 Office Action dated Oct. 3, 2008.
U.S. Appl. No. 10/636,314 Response to Office Action dated Apr. 3, 2009.
U.S. Appl. No. 10/636,314 Notice of Non-Compliant Amendment dated May 4, 2009.
U.S. Appl. No. 10/636,314 Amendment to Office Action Response dated Jun. 4, 2009.
U.S. Appl. No. 10/636,314 Notice of Non-Compliant Amendment dated Jun. 12, 2009.
U.S. Appl. No. 10/636,314 Amendment to Office Action dated Jul. 10, 2009.
U.S. Appl. No. 10/636,314 Final Rejection Office Action dated Oct. 21, 2009.
U.S. Appl. No. 10/636,314 Amendment after Final Office Action dated Dec. 14, 2009.
U.S. Appl. No. 10/636,314 Advisory Action dated Jan. 11, 2010.
U.S. Appl. No. 10/636,314 Notice of Non-Compliant Amendment dated Jan. 11, 2010.
U.S. Appl. No. 10/636,314 Supplemental Amendment and Response dated Mar. 13, 2010.
U.S. Appl. No. 10/636,314 Office Action dated May 11, 2010.
U.S. Appl. No. 10/636,314 Amendment and Response dated Sep. 13, 2010.
U.S. Appl. No. 10/636,314 Final Rejection dated Nov. 24, 2010.
U.S. Appl. No. 10/636,314 Notice of Appeal dated May 19, 2011.
U.S. Appl. No. 10/636,314 Amendment and Request for Continued Examination dated Jul. 19, 2011.
U.S. Appl. No. 10/636,314 Notice of Allowance dated Sep. 13, 2011.
U.S. Appl. No. 10/636,314 Notice of Allowance dated Sep. 19, 2011.
U.S. Appl. No. 10/636,314 Issue Notification dated Oct. 19, 2011.
U.S. Appl. No. 09/474,664, filed Dec. 29, 1999.
U.S. Appl. No. 09/474,664 Office Action dated Sep. 23, 2002.
U.S. Appl. No. 09/474,664 Amendment and Response dated Feb. 24, 2003.
U.S. Appl. No. 09/474,664 Notice of Allowance dated May 20, 2003.
U.S. Appl. No. 90/010,356 Request for Ex Parte Reexamination dated Dec. 15, 2008.
U.S. Appl. No. 90/010,356 Office Action Granting Reexamination dated Jan. 17, 2009.
U.S. Appl. No. 90/010,356 First Office Action dated Jul. 7, 2009.
U.S. Appl. No. 90/010,356 First Office Action Response dated Sep. 1, 2009.
U.S. Appl. No. 90/010,356 Patent Owner Interview Summary dated Oct. 23, 2009.
U.S. Appl. No. 90/010,356 Office Action Final dated Dec. 4, 2009.
U.S. Appl. No. 90/010,356 Amendment and Response to Office Action dated Dec. 18, 2009.
U.S. Appl. No. 90/010,356 Amendment and Response to Office Action dated Jan. 4, 2010.
U.S. Appl. No. 90/010,356 Advisory Action dated Jan. 21, 2010.

(56) **References Cited**

OTHER PUBLICATIONS

U.S. Appl. No. 90/010,356 Amendment and Response to Advisory Action dated Feb. 8, 2010.
U.S. Appl. No. 90/010,356 Notice of Intent to Issue a Reexam Certificate dated Mar. 2, 2010.
U.S. Appl. No. 90/010,356 Reexamination Certificate Issued dated Jun. 22, 2010.
U.S. Appl. No. 95/000,659 Inter Partes Reexam Request dated Feb. 13, 2012.
U.S. Appl. No. 95/000,659 Order Granting Reexamination dated Apr. 3, 2012.
U.S. Appl. No. 95/000,659 Office Action dated Apr. 3, 2012.
U.S. Appl. No. 95/000,659 Office Action Response dated Jun. 4, 2012 (including Exhibits 1 & 2) (4 documents).
U.S. Appl. No. 95/000,659 Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012.
U.S. Appl. No. 95/000,659 Appendix R-1 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Declaration of Prof. Dr. Bernhard Plattner).
U.S. Appl. No. 95/000,659 Appendix R-2 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Prof. Dr. Bernhard Plattner CV).
U.S. Appl. No. 95/000,659 Appendix R-3 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Listing of Publications to Prof. Dr. Bernhard Plattner updated Feb. 2012).
U.S. Appl. No. 95/000,659 Appendix R-4 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012(Office Action Granting Reexamination in 95/000,660 dated May 10, 2012).
U.S. Appl. No. 95/000,659 Appendix R-5 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Office Action in 95/000,660 dated May 10, 2012).
U.S. Appl. No. 95/000,659 Appendix R-6 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Implicit Networks, Inc. U.S. Pat. No. 6,629,163 Claims Chart).
U.S. Appl. No. 95/000,659 Appendix R-7 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Internet Protocol DARPA Internet Program Protocol Specification dated Sep. 1991).
U.S. Appl. No. 95/000,659 Appendix R-8 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Atkinson, IP Encapsulating Security Payload (ESP) dated Aug. 1995).
U.S. Appl. No. 95/000,659 Appendix R-9 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Claim Construction Order dated Feb. 29, 2012).
U.S. Appl. No. 95/000,659 Appendix R-10-1 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (vol. I of Edward Balassanian Deposition Transcript dated May 30, 2012).
U.S. Appl. No. 95/000,659 Appendix R-10-2 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (vol. II of Edward Balassanian Deposition Transcript dated May 31, 2012).
U.S. Appl. No. 95/000,659 Appendix R-10-3 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (vol. III of Edward Balassanian Deposition Transcript dated Jun. 7, 2012).
U.S. Appl. No. 95/000,659 Appendix R-10-4 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (vol. IV of Edward Balassanian Deposition Transcript dated Jun. 8, 2012).
U.S. Appl. No. 95/000,659 Appendix R-11 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Implicit Networks, Inc.'s Response to Juniper Networks, Inc.'s First Set of Requests for Admission 1-32).
U.S. Appl. No. 95/000,659 Action Closing Prosecution dated Oct. 1, 2012.
U.S. Appl. No. 95/000,659 Petition to Withdraw and Reissue Action Closing Prosecution dated Nov. 20, 2012.
U.S. Appl. No. 95/000,659 Patent Owner Comments to Action Closing Prosecution dated Dec. 3, 2012.
U.S. Appl. No. 95/000,659 Opposition to Petition dated Dec. 17, 2012.
U.S. Appl. No. 95/000,659 Third Party Comments to Action Closing Prosecution dated Jan. 2, 2013.
U.S. Appl. No. 95/000,660 Inter Partes Reexam Request dated Mar. 2, 2012.
U.S. Appl. No. 95/000,660 Order Granting Reexamination dated May 10, 2012.
U.S. Appl. No. 95/000,660 Office Action dated May 10, 2012.
U.S. Appl. No. 95/000,660 Response to Office Action dated Jul. 10, 2012 (including Exhibits 1 and 2).
U.S. Appl. No. 95/000,660 Third Party Comments to Office After Patent Owner's Response dated Aug. 8, 2012 (including Revised Comments).
U.S. Appl. No. 95/000,660 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Declaration of Prof. Dr. Bernhard Plattner).
U.S. Appl. No. 95/000,660 Appendix R-1 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Prof. Dr. Bernhard Plattner CV).
U.S. Appl. No. 95/000,660 Appendix R-3 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Listing of Publications to Prof. Dr. Bernhard Plattner updated Feb. 2012).
U.S. Appl. No. 95/000,660 Appendix R-4 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012(Office Action Granting Reexamination in 95/000,660 dated May 10, 2012).
U.S. Appl. No. 95/000,660 Appendix R-5 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Office Action in 95/000,660 dated May 10, 2012).
U.S. Appl. No. 95/000,660 Appendix R-6 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Implicit Networks, Inc. U.S. Pat. No. 6,629,163 Claims Chart).
U.S. Appl. No. 95/000,660 Appendix R-7 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Internet Protocol DARPA Internet Program Protocol Specification dated Sep. 1991).
U.S. Appl. No. 95/000,660 Appendix R-8 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Atkinson, IP Encapsulating Security Payload (ESP) dated Aug. 1995).
U.S. Appl. No. 95/000,660 Appendix R-9 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Claim Construction Order dated Feb. 29, 2012).
U.S. Appl. No. 95/000,660 Appendix R-10 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (vol. I-IV of Edward Balassanian Deposition Transcript dated May 30, 2012).
U.S. Appl. No. 95/000,660 Appendix R-11 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Shacham, A., et al, "*IP Payload Compression Protocol*", Network Working Group, RFC 3173 Sep. 2001).
U.S. Appl. No. 95/000,660 Appendix R-12 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Shacham, A., et al, "*IP Payload Compression Protocol*", Network Working Group, RFC 2393 Dec. 1998).
U.S. Appl. No. 95/000,660 Appendix R-13 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (' 163 Pfeiffer Claim Chart).
U.S. Appl. No. 95/000,660 Appendix R-14 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Ylonen, T., "*SSH Transport Layer Protocol*", Network Working Group—Draft Feb. 22, 1999).
U.S. Appl. No. 95/000,660 Appendix R-15 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Dommety, G., "*Key and Sequence Number Extensions to GRE*", Network Working Group, RFC 2890 Sep. 2000).

(56) **References Cited**

OTHER PUBLICATIONS

U.S. Appl. No. 95/000,660 Appendix R-16 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Monsour, R., et al, "*Compression in IP Security*" Mar. 1997).
U.S. Appl. No. 95/000,660 Appendix R-17 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Friend, R., Internet Working Group RFC 3943 dated Nov. 2004 *Transport Layer Security Protocol Compression Using Lempel-Ziv-Stac*).
U.S. Appl. No. 95/000,660 Appendix R-18 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012(Implicit Networks, Inc.'s Response to Juniper Networks, Inc.'s First Set of Requests for Admission 1-32).
U.S. Appl. No. 95/000,660 Revised—Third Party Comments to Office After Patent Owner's Response dated Nov. 2, 2012.
U.S. Appl. No. 95/000,660 Action Closing Prosecution dated Dec. 21, 2012.
U.S. Appl. No. 95/000,660 Comments to Action Closing Prosecution dated Feb. 21, 2013 (including Dec of Dr. Ng).
U.S. Appl. No. 95/000,660 Third Party Comments to Action Closing Prosecution dated Mar. 25, 2013.
PCT/US00/33634—PCT application (WO 01/2077 A2—Jul. 12, 2001).
PCT/US00/33634—Written Opinion (WO 01/50277 A3—dated Feb. 14, 2002).
PCT/US00/33634—International Search Report (dated Oct. 9, 2001).
PCT/US00/33634—Response to Official Communication dated Dec. 7, 2001 (dated Mar. 21, 2002).
PCT/US00/33634—International Preliminary Examination Report (dated Apr. 8, 2002).
PCT/US00/33634—Official Communication (dated Jan. 24, 2003).
PCT/US00/33634—Response to Official Communication dated Jan. 24, 2003 (dated Mar. 12, 2003).
PCT/US00/33634—Official Communication (dated May 13, 2004).
PCT/US00/33634—Response to Summons to Attend Oral Proceeding dated May 13, 2004 (dated Oct. 9, 2004).
PCT/US00/33634—Decision to Refuse a European Patent application (dated Nov. 12, 2004).
PCT/US00/33634—Minutes of the oral proceedings before the Examining Division (dated Oct. 12, 2004).
PCT/US00/33634—Closure of the procedure in respect to Application No. 00984234.5-2212 (dated Feb. 22, 2005).
May 3, 2013 Expert Report of Dr. Alfonso Cardenas Regarding Validity of U.S. Pat. Nos. 6,877,006; 7,167,864; 7,720,861; and 8,082,268 (6 documents).
Expert Report of Dr. Alfonso Cardenas Regarding Validity of U.S. Pat. No. 7,167,864 (3 documents).
"InfoReports User Guide: Version 3.3.1;" Platinum Technology, Publication No. PRO-X-331-UG00-00, printed Apr. 1998; pp. 1-430.

* cited by examiner

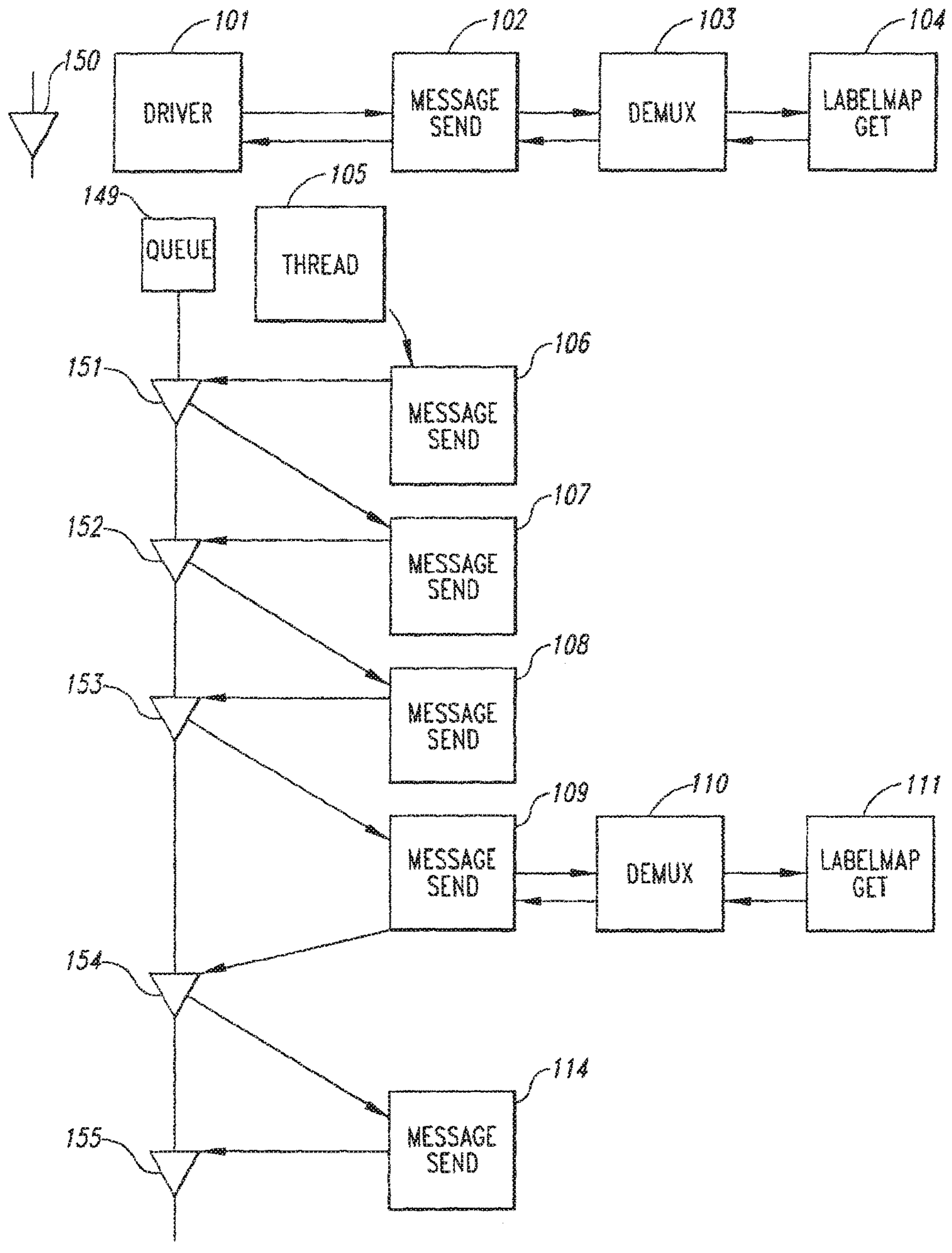

101
DRIVER
102
MESSAGE SEND
103
DEMUX
104
LABELMAP GET
150
149
QUEUE
105
THREAD
106
MESSAGE SEND
151
107
MESSAGE SEND
152
108
MESSAGE SEND
153
109
MESSAGE SEND
110
DEMUX
111
LABELMAP GET
154
114
MESSAGE SEND
155

Fig. 1

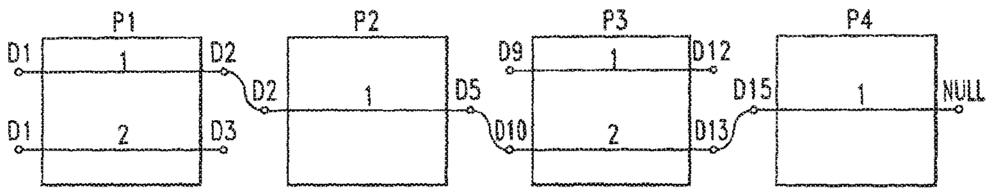

P1
P2
P3
P4
D1
1
D2
D2
1
D5
D9
1
D12
D15
1
NULL
D1
2
D3
D10
2
D13

Fig. 2

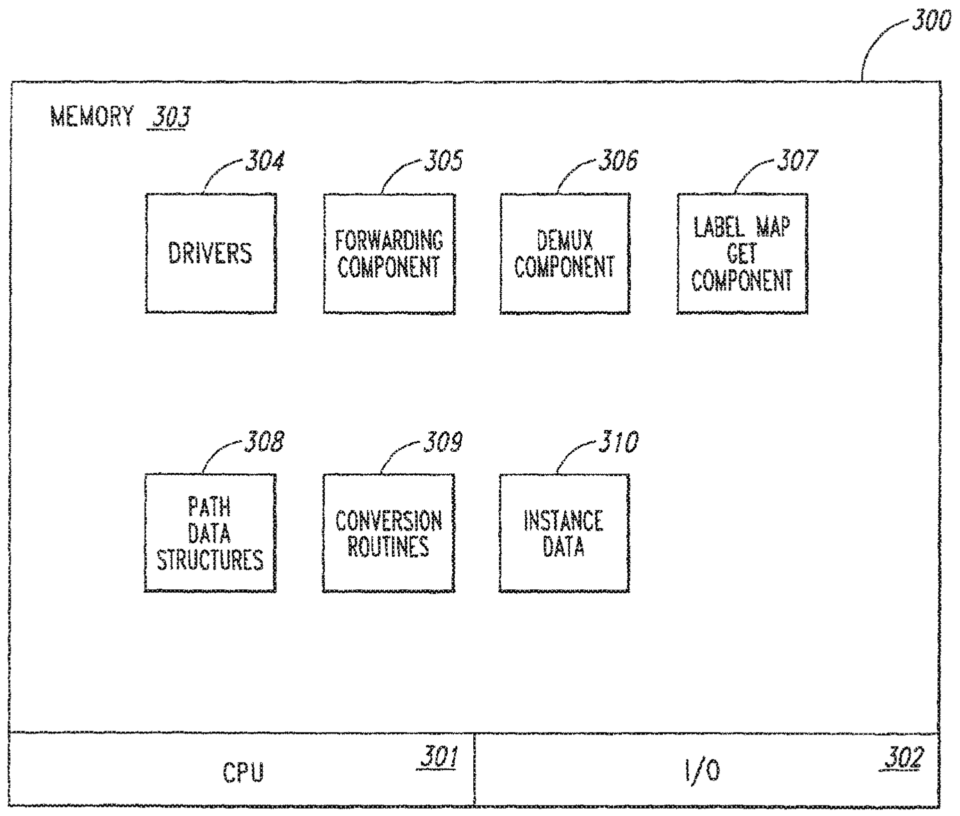

300
MEMORY 303
304
DRIVERS
305
FORWARDING COMPONENT
306
DEMUX COMPONENT
307
LABEL MAP GET COMPONENT
308
PATH DATA STRUCTURES
309
CONVERSION ROUTINES
310
INSTANCE DATA
CPU 301
I/O 302

Fig. 3

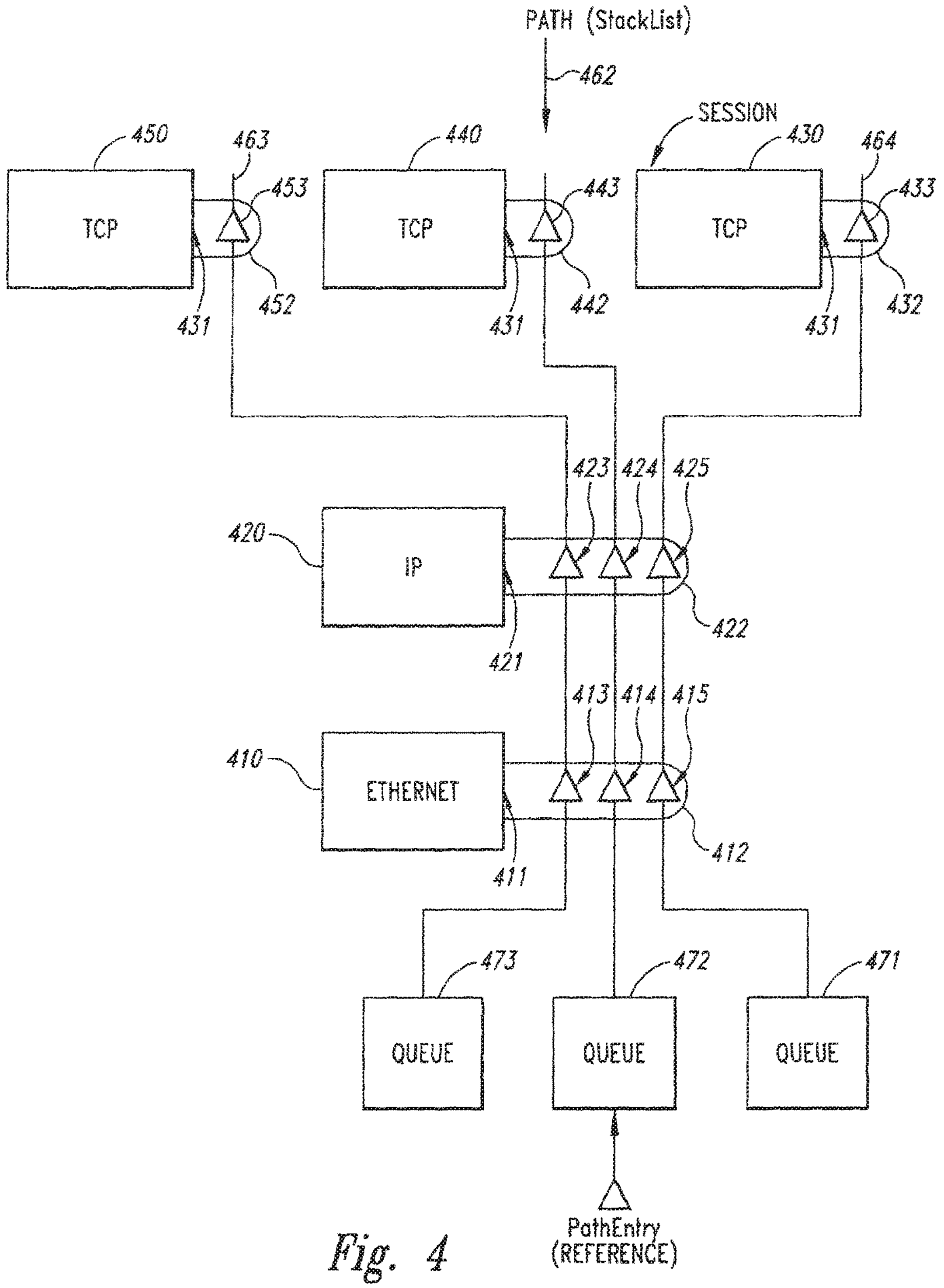
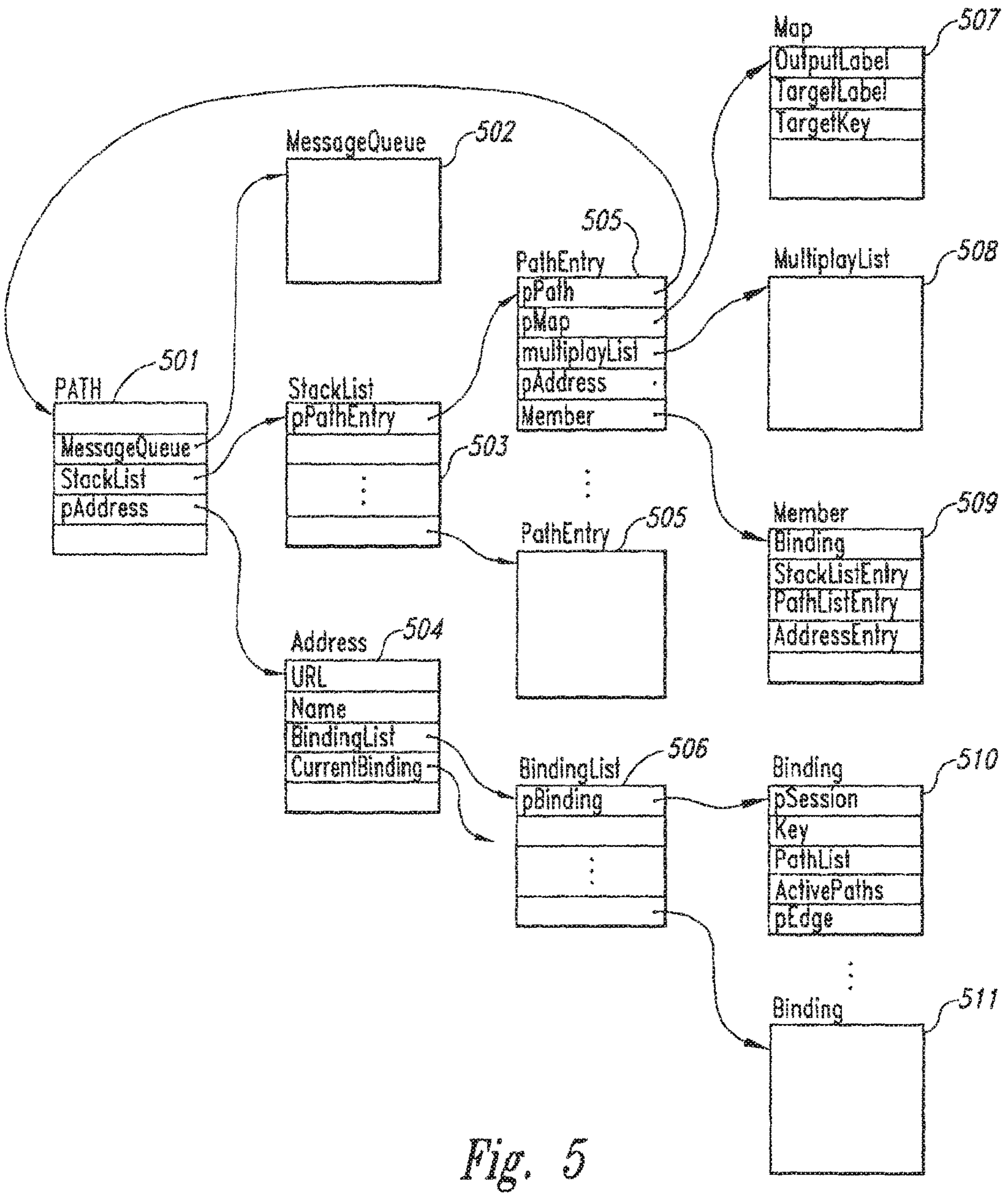

PATH (StackList)
462
SESSION
450
463
453
440
443
430
464
433
TCP
TCP
TCP
452
431
431
442
431
432
423
424
425
420
IP
422
421
413
414
415
410
ETHERNET
412
411
473
472
471
QUEUE
QUEUE
QUEUE
PathEntry
(REFERENCE)

Fig. 4



Map
507
OutputLabel
TargetLabel
TargetKey
MessageQueue
502
505
PathEntry
pPath
pMap
multiplayList
pAddress
Member
MultiplayList
508
501
PATH
MessageQueue
StackList
pAddress
StackList
pPathEntry
503
PathEntry
505
Member
509
Binding
StackListEntry
PathListEntry
AddressEntry
Address
504
URL
Name
BindingList
CurrentBinding
BindingList
506
pBinding
Binding
510
pSession
Key
PathList
ActivePaths
pEdge
Binding
511

Fig. 5

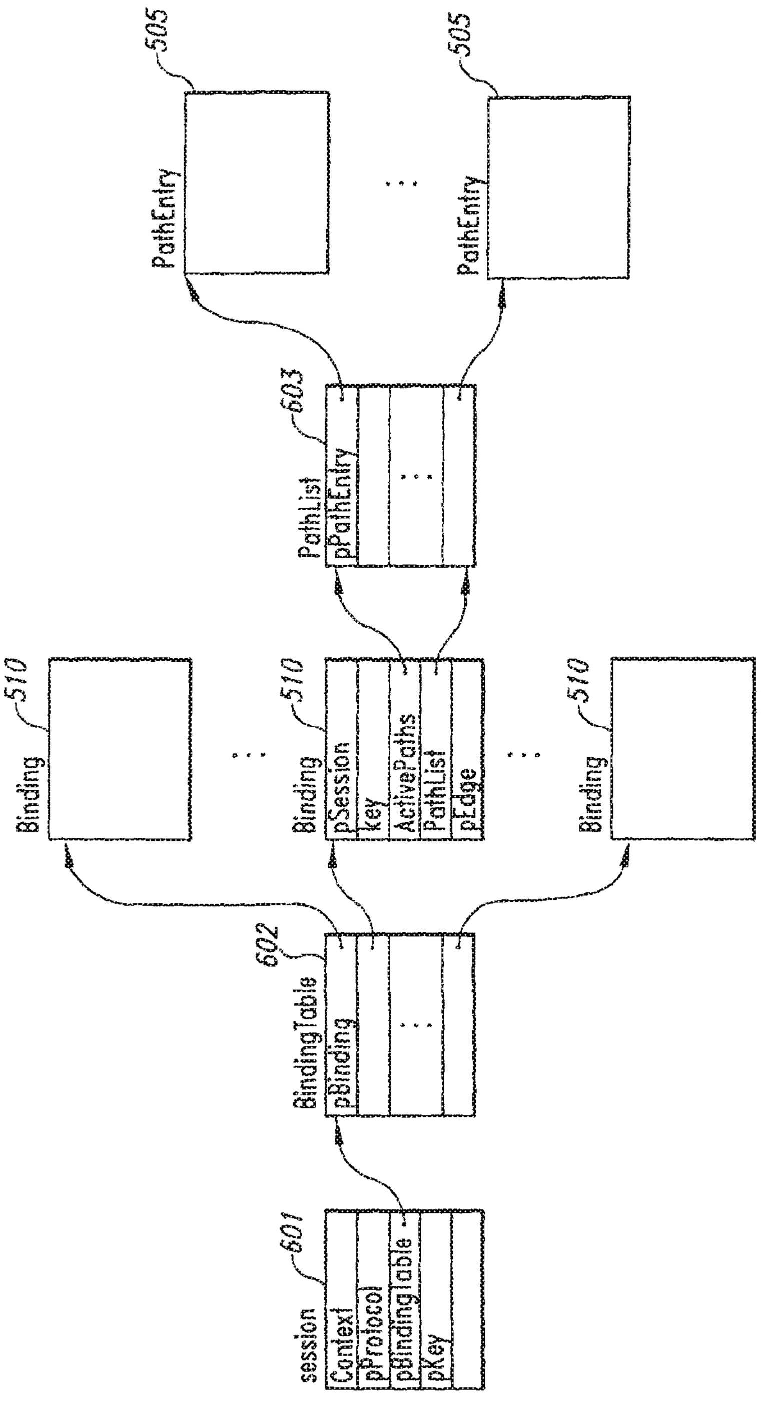

505
PathEntry
505
PathEntry
603
PathList
pPathEntry
510
Binding
510
Binding
pSession
key
ActivePaths
PathList
pEdge
510
Binding
602
BindingTable
pBinding
601
session
Context
pProtocol
pBindingTable
pKey

Fig. 6

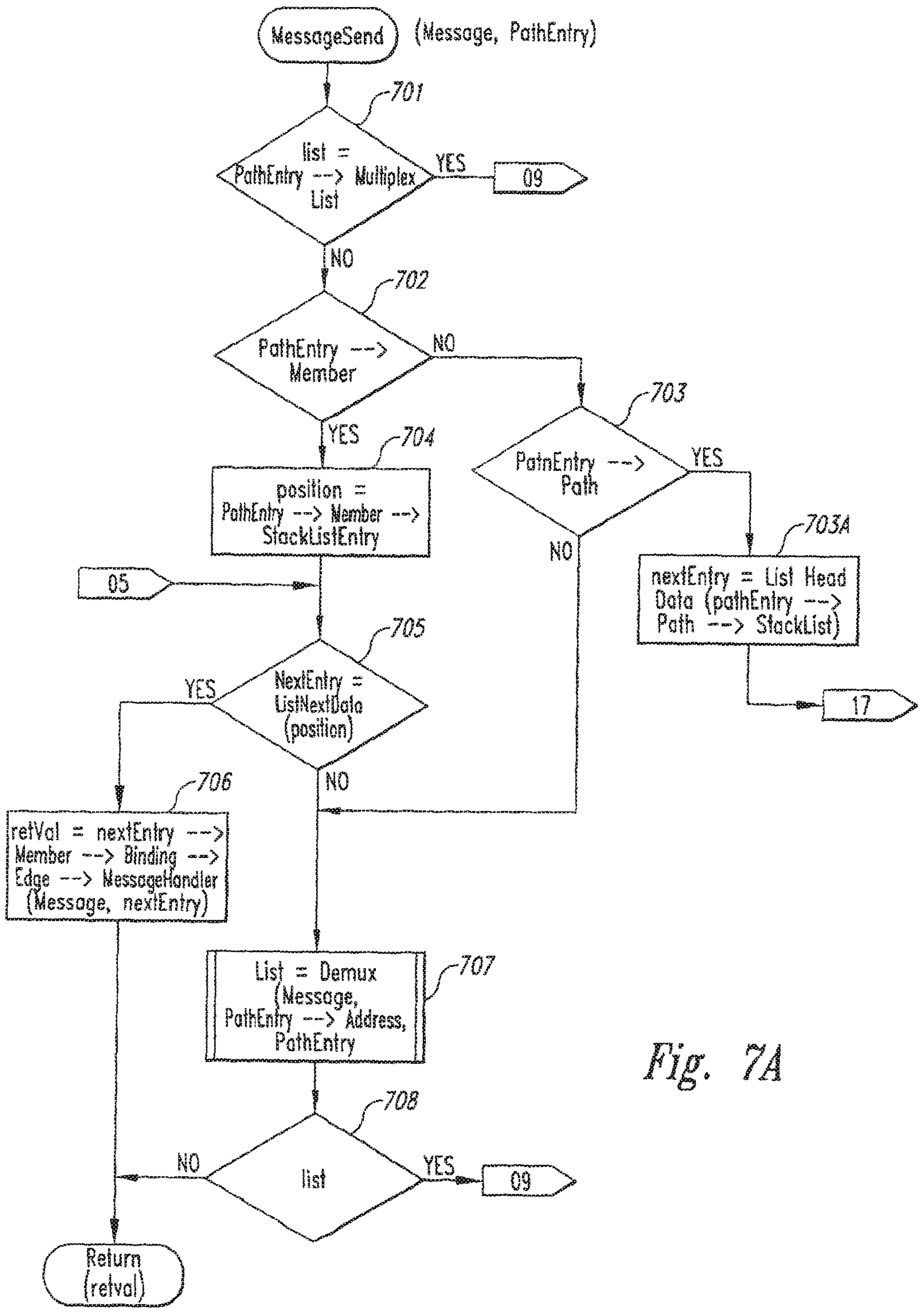

MessageSend
(Message, PathEntry)
701
list = PathEntry --> Multiplex List
YES
09
NO
702
PathEntry --> Member
NO
703
PatnEntry --> Path
YES
703A
nextEntry = List Head Data (pathEntry --> Path --> StackList)
17
NO
YES
704
position = PathEntry --> Member --> StackListEntry
05
705
NextEntry = ListNextData (position)
YES
NO
706
retVal = nextEntry --> Member --> Binding --> Edge --> MessageHandler (Message, nextEntry)
707
List = Demux (Message, PathEntry --> Address, PathEntry
708
list
NO
YES
09
Return (retval)

Fig. 7A

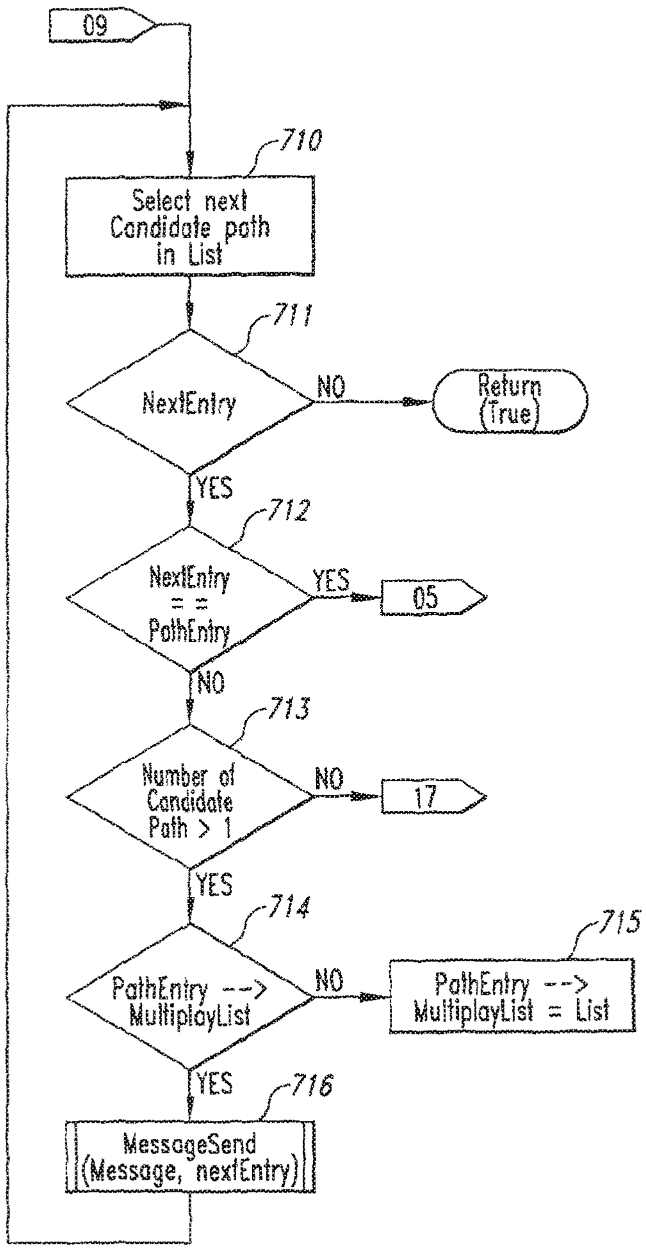

09
710
Select next Candidate path in List
711
NextEntry
NO
Return (True)
YES
712
NextEntry == PathEntry
YES
05
NO
713
Number of Candidate Path > 1
NO
17
YES
714
PathEntry --> MultiplayList
NO
715
PathEntry --> MultiplayList = List
YES
716
MessageSend (Message, nextEntry)

Fig. 7B

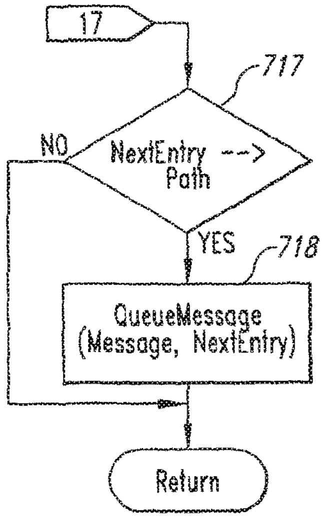

17
717
NextEntry --> Path
NO
YES
718
QueueMessage (Message, NextEntry)
Return

Fig. 7C

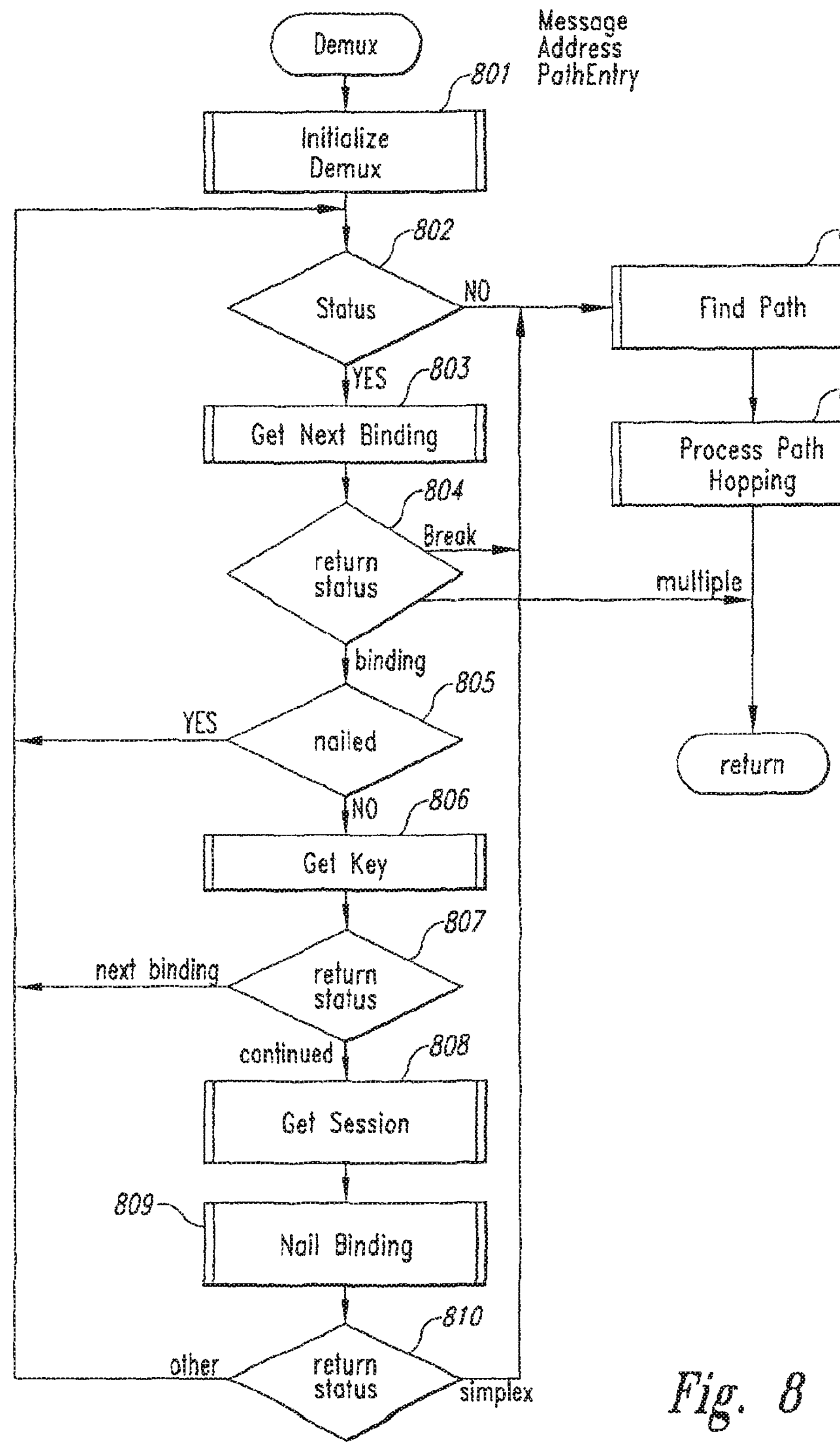

Demux
Message
Address
PathEntry
801
Initialize Demux
802
Status
NO
YES
811
Find Path
812
Process Path Hopping
803
Get Next Binding
804
return status
Break
multiple
binding
805
nailed
YES
NO
return
806
Get Key
807
return status
next binding
continued
808
Get Session
809
Nail Binding
810
return status
other
simplex

Fig. 8

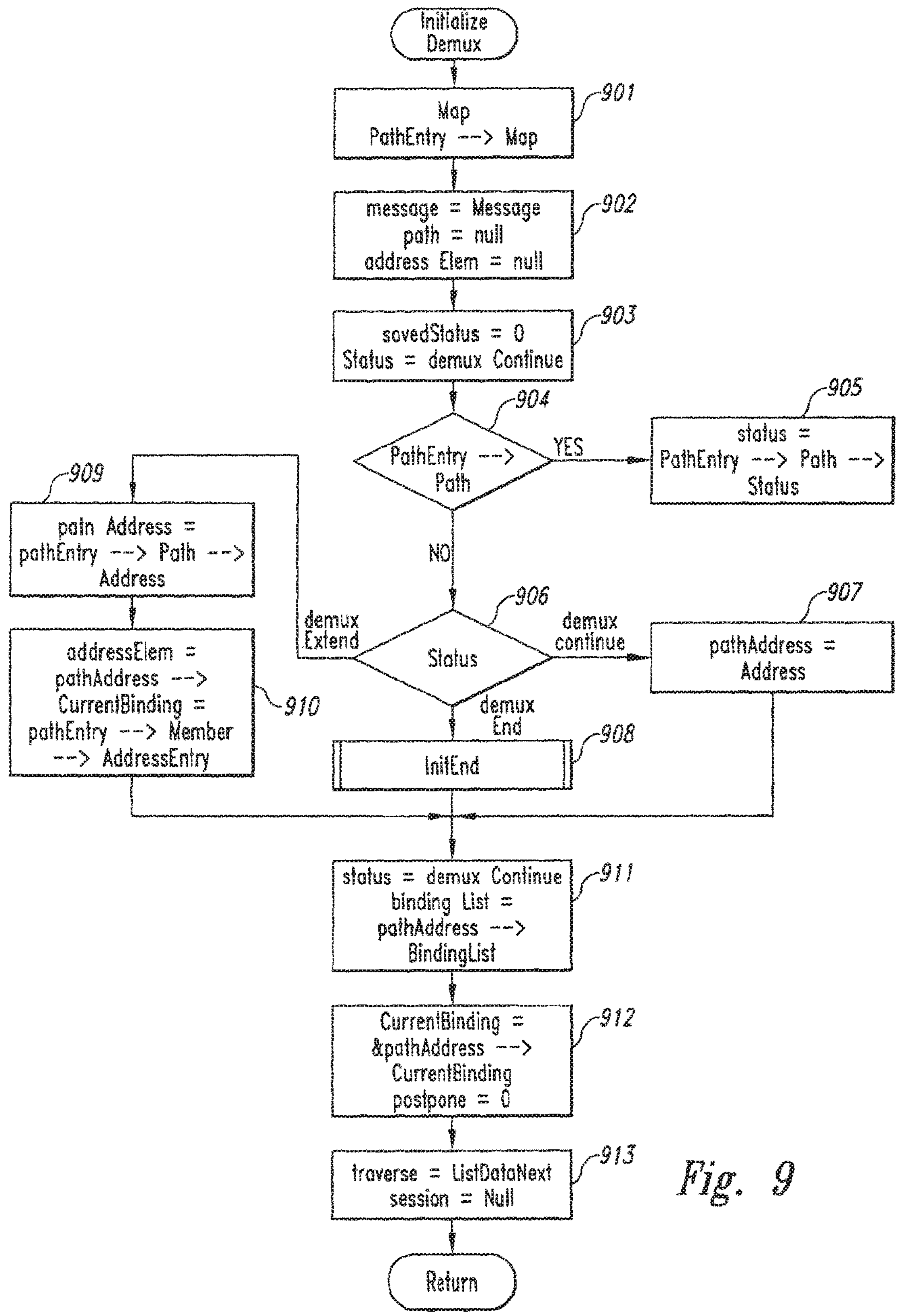

Initialize Demux
901
Map PathEntry --> Map
902
message = Message path = null address Elem = null
903
savedStatus = 0 Status = demux Continue
904
PathEntry --> Path
YES
905
status = PathEntry --> Path --> Status
NO
906
Status
demux Extend
demux continue
demux End
907
pathAddress = Address
908
InitEnd
909
path Address = pathEntry --> Path --> Address
910
addressElem = pathAddress --> CurrentBinding = pathEntry --> Member --> AddressEntry
911
status = demux Continue binding List = pathAddress --> BindingList
912
CurrentBinding = &pathAddress --> CurrentBinding postpone = 0
913
traverse = ListDataNext session = Null
Return

Fig. 9

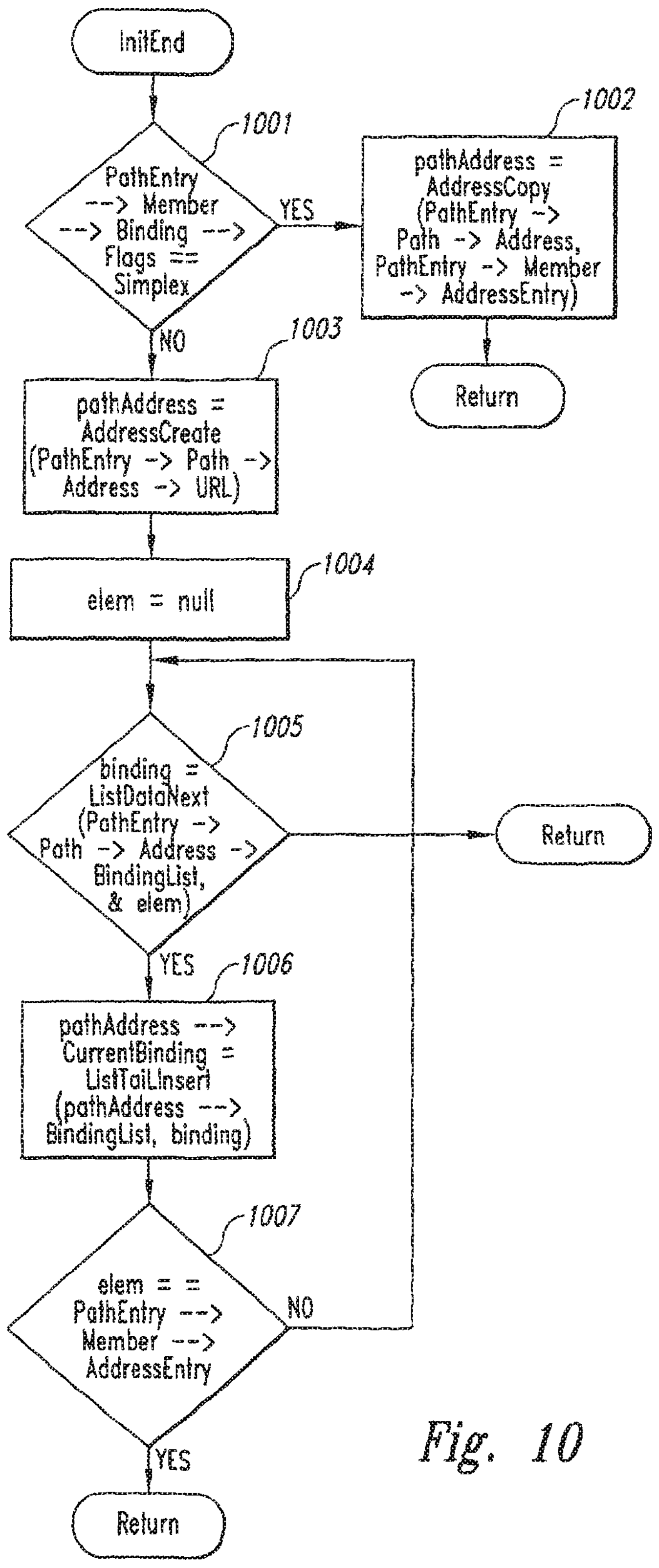
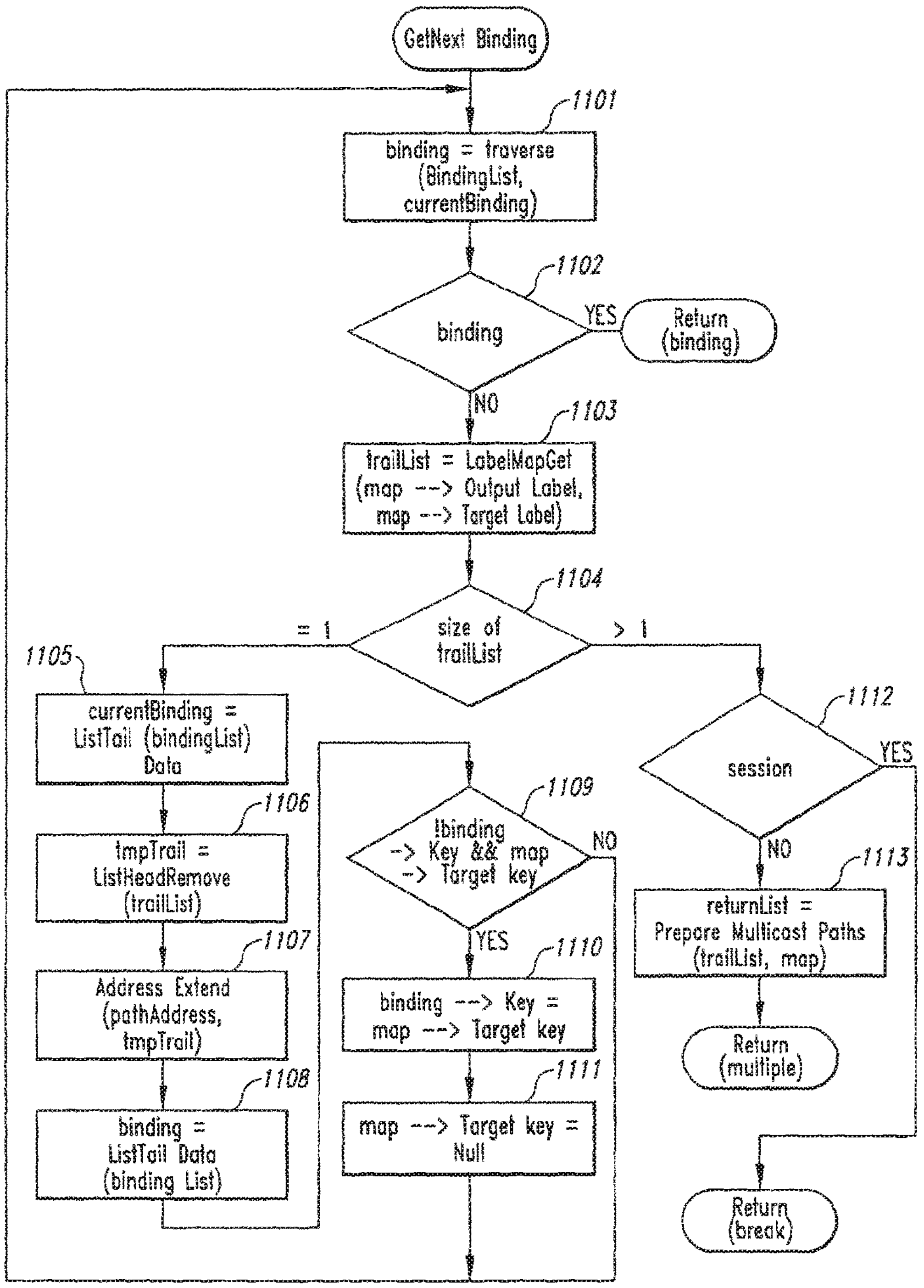

InitEnd
1001
PathEntry --> Member --> Binding --> Flags == Simplex
YES
1002
pathAddress = AddressCopy (PathEntry -> Path -> Address, PathEntry -> Member -> AddressEntry)
Return
NO
1003
pathAddress = AddressCreate (PathEntry -> Path -> Address -> URL)
1004
elem = null
1005
binding = ListDataNext (PathEntry -> Path -> Address -> BindingList, & elem)
Return
YES
1006
pathAddress --> CurrentBinding = ListTailInsert (pathAddress --> BindingList, binding)
1007
elem = = PathEntry --> Member --> AddressEntry
NO
YES
Return

Fig. 10



GetNext Binding
1101
binding = traverse (BindingList, currentBinding)
1102
binding
YES
Return (binding)
NO
1103
trailList = LabelMapGet (map --> Output Label, map --> Target Label)
1104
size of trailList
= 1
> 1
1105
currentBinding = ListTail (bindingList) Data
1106
tmpTrail = ListHeadRemove (trailList)
1107
Address Extend (pathAddress, tmpTrail)
1108
binding = ListTail Data (binding List)
1109
!binding -> Key && map -> Target key
NO
YES
1110
binding --> Key = map --> Target key
1111
map --> Target key = Null
1112
session
YES
NO
1113
returnList = Prepare Multicast Paths (trailList, map)
Return (multiple)
Return (break)

Fig. 11

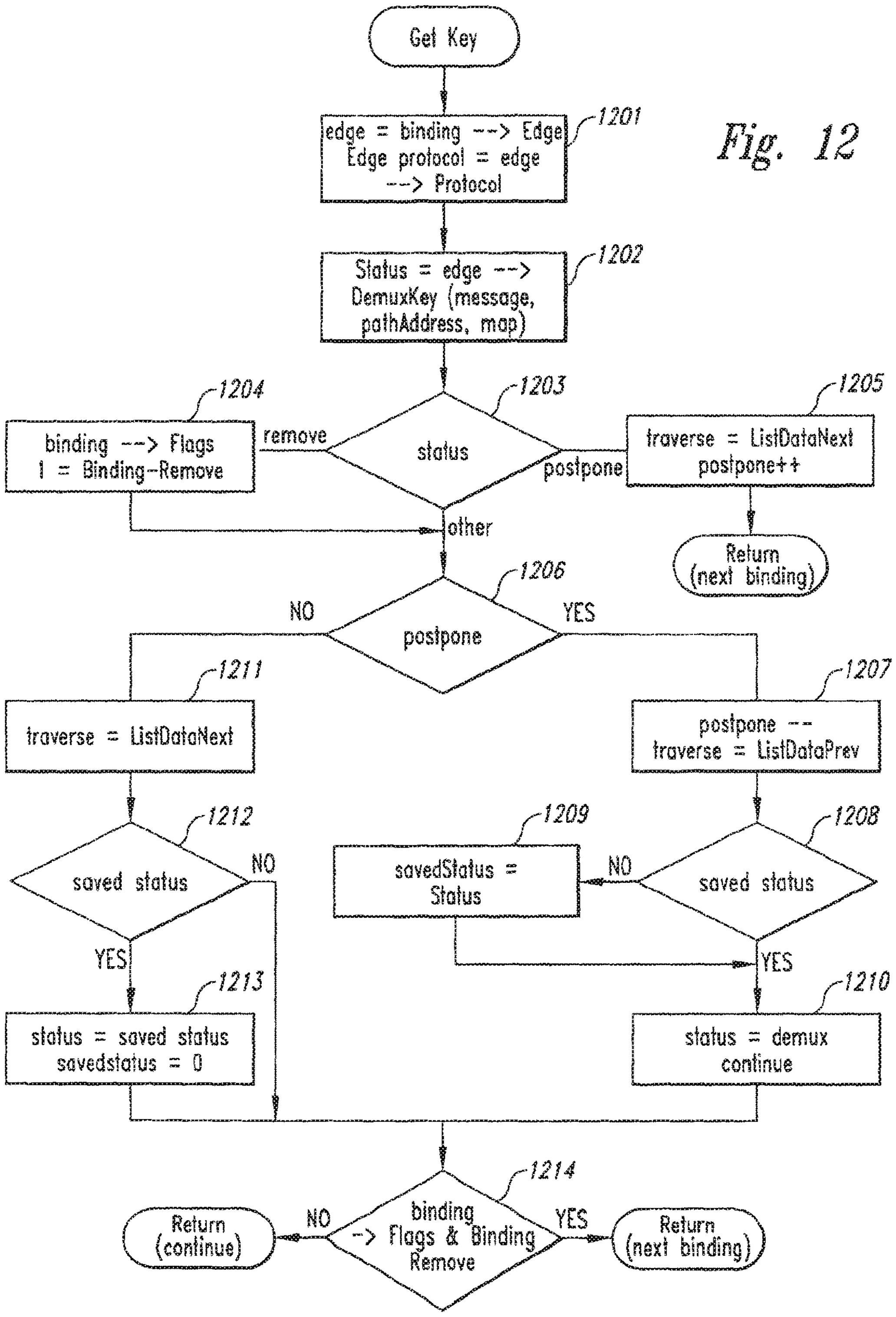

Get Key
edge = binding --> Edge
Edge protocol = edge --> Protocol
1201
Status = edge --> DemuxKey (message, pathAddress, map)
1202
1203
status
remove
postpone
other
1204
binding --> Flags
1 = Binding-Remove
1205
traverse = ListDataNext
postpone++
Return (next binding)
1206
postpone
NO
YES
1211
traverse = ListDataNext
1207
postpone --
traverse = ListDataPrev
1212
saved status
NO
YES
1209
savedStatus = Status
1208
saved status
NO
YES
1213
status = saved status
savedstatus = 0
1210
status = demux
continue
1214
binding -> Flags & Binding Remove
NO
YES
Return (continue)
Return (next binding)

Fig. 12

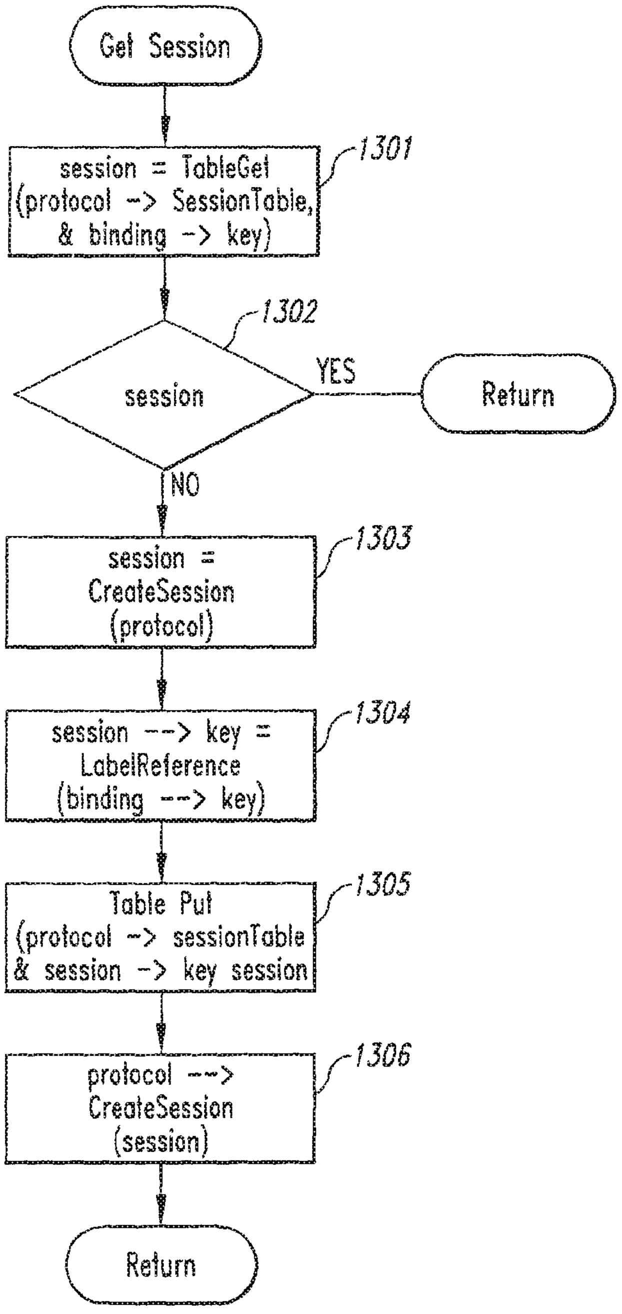

Get Session
session = TableGet (protocol -> SessionTable, & binding -> key)
1301
session
1302
YES
Return
NO
session = CreateSession (protocol)
1303
session --> key = LabelReference (binding --> key)
1304
Table Put (protocol -> sessionTable & session -> key session
1305
protocol --> CreateSession (session)
1306
Return

Fig. 13

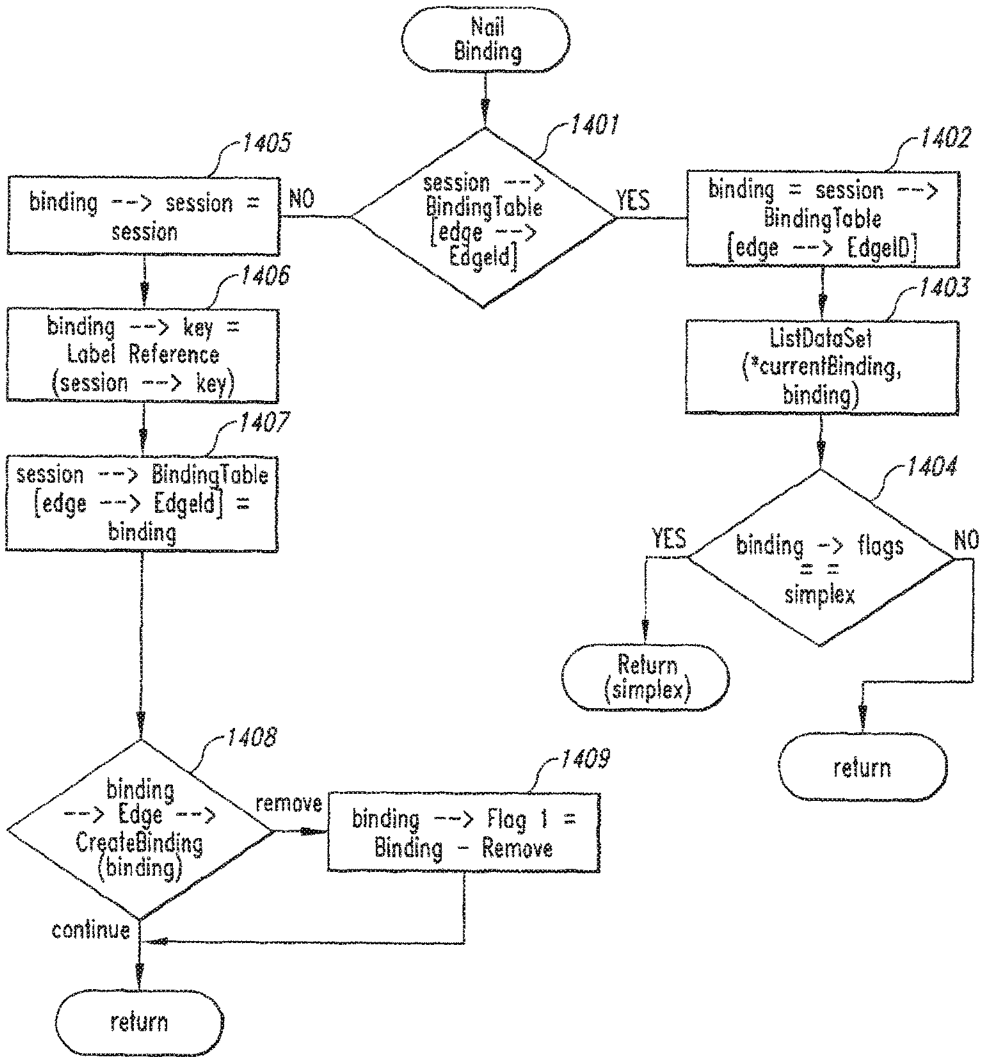

Nail Binding
1401
session --> BindingTable [edge --> EdgeId]
YES
NO
1402
binding = session --> BindingTable [edge --> EdgeID]
1403
ListDataSet (*currentBinding, binding)
1404
binding -> flags = = simplex
YES
NO
Return (simplex)
return
1405
binding --> session = session
1406
binding --> key = Label Reference (session --> key)
1407
session --> BindingTable [edge --> EdgeId] = binding
1408
binding --> Edge --> CreateBinding (binding)
remove
1409
binding --> Flag 1 = Binding - Remove
continue
return

Fig. 14

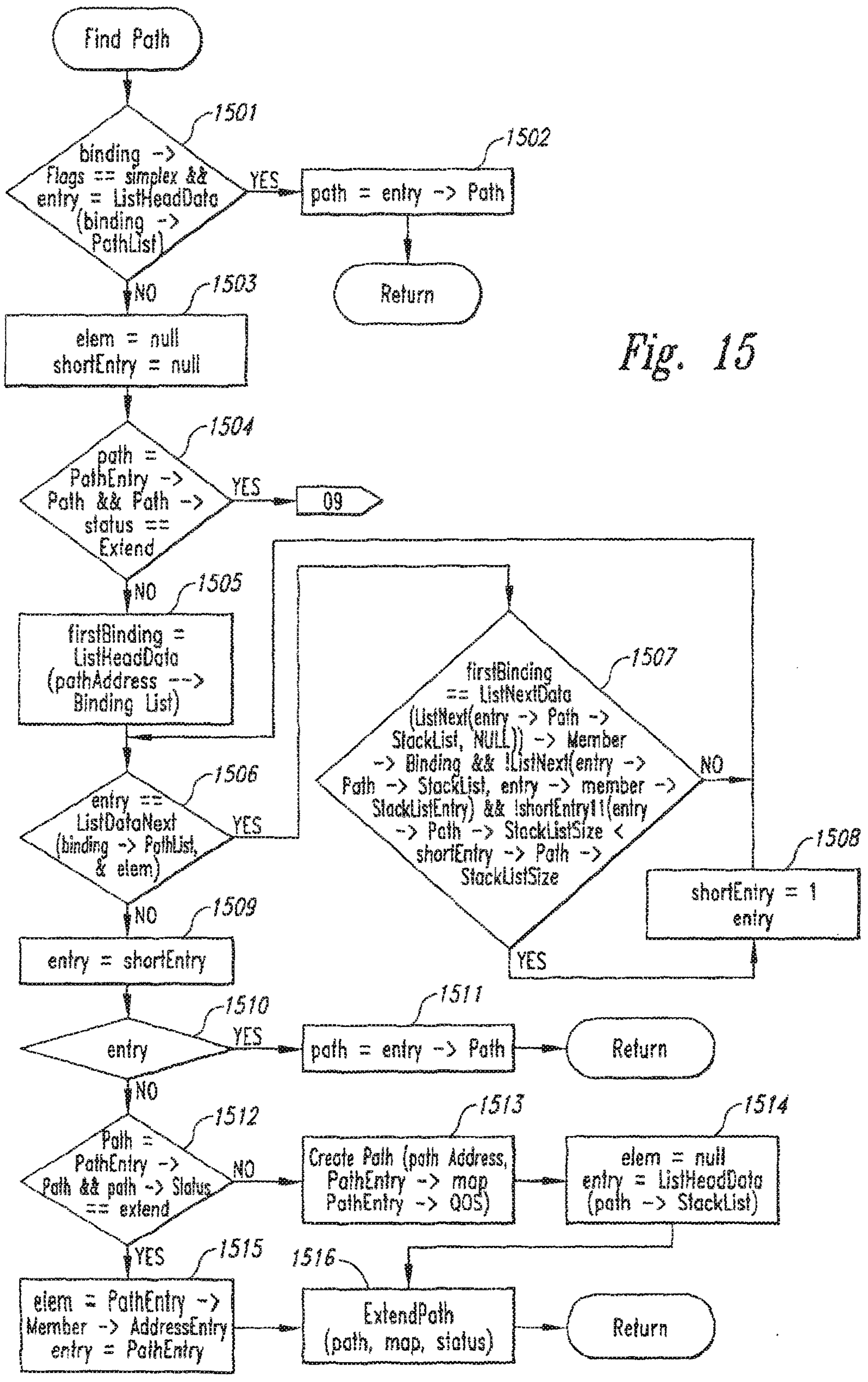

Find Path
1501
binding -> Flags == simplex && entry = ListHeadData (binding -> PathList)
YES
1502
path = entry -> Path
Return
NO
1503
elem = null
shortEntry = null
1504
path = PathEntry -> Path && Path -> status == Extend
YES
09
NO
1505
firstBinding = ListHeadData (pathAddress --> Binding List)
1506
entry == ListDataNext (binding -> PathList, & elem)
YES
1507
firstBinding == ListNextData (ListNext(entry -> Path -> StackList, NULL)) -> Member -> Binding && !ListNext(entry -> Path -> StackList, entry -> member -> StackListEntry) && !shortEntry11(entry -> Path -> StackListSize < shortEntry -> Path -> StackListSize
NO
YES
1508
shortEntry = 1 entry
NO
1509
entry = shortEntry
1510
entry
YES
1511
path = entry -> Path
Return
NO
1512
Path = PathEntry -> Path && path -> Status == extend
NO
1513
Create Path (path Address, PathEntry -> map PathEntry -> QOS)
1514
elem = null
entry = ListHeadData (path -> StackList)
YES
1515
elem = PathEntry -> Member -> AddressEntry
entry = PathEntry
1516
ExtendPath (path, map, status)
Return

Fig. 15

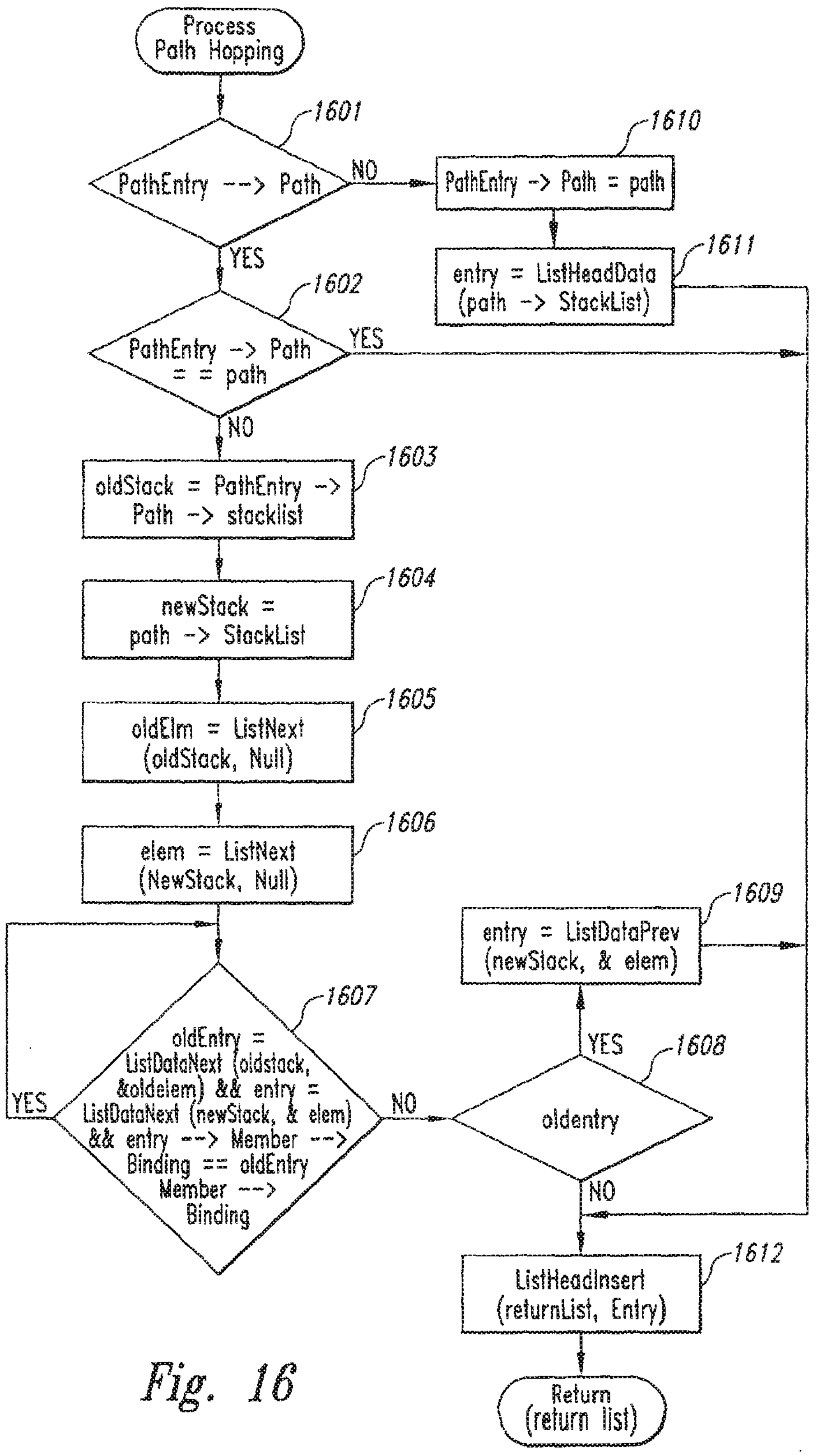

Process Path Hopping
1601
PathEntry --> Path
NO
1610
PathEntry -> Path = path
1611
entry = ListHeadData (path -> StackList)
YES
1602
PathEntry -> Path = = path
YES
NO
1603
oldStack = PathEntry -> Path -> stacklist
1604
newStack = path -> StackList
1605
oldElm = ListNext (oldStack, Null)
1606
elem = ListNext (NewStack, Null)
1609
entry = ListDataPrev (newStack, & elem)
1607
oldEntry = ListDataNext (oldstack, &oldelem) && entry = ListDataNext (newStack, & elem) && entry --> Member --> Binding == oldEntry Member --> Binding
YES
NO
YES
1608
oldentry
NO
1612
ListHeadInsert (returnList, Entry)
Return (return list)

Fig. 16

METHOD AND SYSTEM FOR DATA DEMULTIPLEXING

CROSS REFERENCES TO RELATED APPLICATIONS

The present application is a continuation of U.S. application Ser. No. 15/050,027, filed Feb. 22, 2016 (now U.S. Pat. No. 9,591,104), which is a continuation of U.S. application Ser. No. 14/230,952, filed Mar. 31, 2014 (now U.S. Pat. No. 9,270,790), which is a continuation of U.S. Appl. Ser. No. 13/911,324, filed Jun. 6, 2013 (now U.S. Pat. No. 8,694,683), which is a continuation of U.S. application Ser. No. 13/236,090, filed Sep. 19, 2011 know abandoned), which is a continuation of U.S. application Ser. No. 10/636,314, filed Aug. 6, 2003 know U.S. Pat. No. 8,055,786), which is a continuation of U.S. Appl. Ser. No. 09/474,664, filed Dec. 29, 1999 (now U.S. Pat. No. 6,629,163); the disclosures of each of the above-referenced applications are incorporated by reference herein in their entireties.

TECHNICAL FIELD

The present invention relates generally to a computer system for data demultiplexing.

BACKGROUND

Computer systems, which are becoming increasingly pervasive, generate data in a wide variety of formats. The Internet is an example of interconnected computer systems that generate data in many different formats. Indeed, when data is generated on one computer system and is transmitted to another computer system to be displayed, the data may be converted in many different intermediate formats before it is eventually displayed. For example, the generating computer system may initially store the data in a bitmap format. To send the data to another computer system, the computer system may first compress the bitmap data and then encrypt the compressed data. The computer system may then convert that compressed data into a TCP format and then into an IP format. The IP formatted data may be converted into a transmission format, such as an ethernet format. The data in the transmission format is then sent to a receiving computer system. The receiving computer system would need to perform each of these conversions in reverse order to convert the data in the bitmap format. In addition, the receiving computer system may need to convert the bitmap data into a format that is appropriate for rendering on output device.

In order to process data in such a wide variety of formats, both sending and receiving computer systems need to have many conversion routines available to support the various formats. These computer systems typically use predefined configuration information to load the correct combination of conversion routines for processing data. These computer systems also use a process-oriented approach when processing data with these conversion routines. When using a process-oriented approach, a computer system may create a separate process for each conversion that needs to take place. A computer system in certain situations, however, can be expected to receive data and to provide data in many different formats that may not be known until the data is received. The overhead of statically providing each possible series of conversion routines is very high. For example, a computer system that serves as a central controller for data received within a home would be expected to process data

received via telephone lines, cable TV lines, and satellite connections in many different formats. The central controller would be expected to output the data to computer displays, television displays, entertainment centers, speakers, recording devices, and so on in many different formats. Moreover, since the various conversion routines may be developed by different organizations, it may not be easy to identify that the output format of one conversion routine is compatible with the input format of another conversion routine.

It would be desirable to have a technique for dynamically identifying a series of conversion routines for processing data. In addition, it would be desirable to have a technique in which the output format of one conversion routine can be identified as being compatible with the input format of another conversion routine. It would also be desirable to store the identification of a series of conversion routines so that the series can be quickly identified when data is received.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a block diagram illustrating example processing of a message by the conversion system.

FIG. **2** is a block diagram illustrating a sequence of edges.

FIG. **3** is a block diagram illustrating components of the conversion system in one embodiment.

FIG. **4** is a block diagram illustrating example path data structures in one embodiment.

FIG. **5** is a block diagram that illustrates the interrelationship of the data structures of a path.

FIG. **6** is a block diagram that illustrates the interrelationship of the data structures associated with a session.

FIGS. **7**A, **7**B, and **7**C comprise a flow diagram illustrating the processing of the message send routine.

FIG. **8** is a flow diagram of the demux routine.

FIG. **9** is a flow diagram of the initialize demux routine.

FIG. **10** is a flow diagram of the init end routine.

FIG. **11** is a flow diagram of a routine to get the next binding.

FIG. **12** is a flow diagram of the get key routine.

FIG. **13** is a flow diagram of the get session routine.

FIG. **14** is a flow diagram of the nail binding routine.

FIG. **15** is a flow diagram of the find path routine.

FIG. **16** is a flow diagram of the process of path hopping routine.

DETAILED DESCRIPTION

A method and system for converting a message that may contain multiple packets from an source format into a target format. When a packet of a message is received, the conversion system in one embodiment searches for and identifies a sequence of conversion routines (or more generally message handlers) for processing the packets of the message by comparing the input and output formats of the conversion routines. (A message is a collection of data that is related in some way, such as stream of video or audio data or an email message.) The identified sequence of conversion routines is used to convert the message from the source format to the target format using various intermediate formats. The conversion system then queues the packet for processing by the identified sequence of conversion routines. The conversion system stores the identified sequence so that the sequence can be quickly found (without searching) when the next packet in the message is received. When subsequent packets of the message are received, the conversion system identifies the sequence and queues the packets for pressing by the sequence. Because the conversion system receives multiple messages with different source and target formats and identifies a sequence of conversion routines for each message, the conversion systems effectively "demultiplexes" the messages. That is, the conversion system demultiplexes the messages by receiving the message, identifying the sequence of conversion routines, and controlling the processing of each message by the identified sequence. Moreover, since the conversion routines may need to retain state information between the receipt of one packet of a message and the next packet of that message, the conversion system maintains state information as an instance or session of the conversion routine. The conversion system routes all packets for a message through the same session of each conversion routine so that the same state or instance information can be used by all packets of the message. A sequence of sessions of conversion routines is referred to as a "path." In one embodiment, each path has a path thread associated with it for processing of each packet destined for that path.

In one embodiment, the packets of the messages are initially received by "drivers," such as an Ethernet driver. When a driver receives a packet, it forwards the packet to a forwarding component of the conversion system. The forwarding component is responsible for identifying the session of the conversion routine that should next process the packet and invoking that conversion routine. When invoked by a driver, the forwarding component may use a demultiplexing ("demux") component to identify the session of the first conversion routine of the path that is to process the packet and then queues the packet for processing by the path. A path thread is associated with each path. Each path thread is responsible for retrieving packets from the queue of its path and forwarding the packets to the forwarding component. When the forwarding component is invoked by a path thread, it initially invokes the first conversion routine in the path. That conversion routine processes the packet and forwards the processed packet to the forwarding component, which then invokes the second conversion routine in the path. The process of invoking the conversion routines and forwarding the processed packet to the next conversion routine continues until the last conversion routine in the path is invoked. A conversion routine may defer invocation of the forwarding component until it aggregates multiple packets or may invoke the forwarding component multiple times for a packet once for each sub-packet.

The forwarding component identifies the next conversion routine in the path using the demux component and stores that identification so that the forwarding component can quickly identify the conversion routine when subsequent packets of the same message are received. The demux component, searches for the conversion routine and session that is to next process a packet. The demux component then stores the identification of the session and conversion routine as part of a path data structure so that the conversion system does not need to search for the session and conversion routine when requested to demultiplex subsequent packets of the same message. When searching for the next conversion routine, the demux component invokes a label map get component that identifies the next conversion routine. Once the conversion routine is found, the demux component identifies the session associated with that message by, in one embodiment, invoking code associated with the conversion routine. In general, the code of the conversion routine determines what session should be associated with a message. In certain situations, multiple messages may share the same session. The demux component then extends the path for processing that packet to include that session

and conversion routine. The sessions are identified so that each packet is associated with the appropriate state information. The dynamic identification of conversion routines is described in U.S. patent application Ser. No. 11/933,093, filed on Oct. 31, 2007 (now U.S. Pat. No. 7,730,211), entitled "Method and System for Generating a Mapping Between Types of Data," which is hereby incorporated by reference.

FIG. **1** is a block diagram illustrating example processing of a message by the conversion system. The driver **101** receives the packets of the message from a network. The driver performs any appropriate processing of the packet and invokes a message send routine passing the processed packet along with a reference path entry **150**. The message send routine is an embodiment of the forwarding component. A path is represented by a series of path entries, which are represented by triangles. Each member path entry represents a session and conversion routine of the path, and a reference path entry represents the overall path. The passed reference path entry **150** indicates to the message send routine that it is being invoked by a driver. The message send routine invokes the demux routine **102** to search for and identify the path of sessions that is to process the packet. The demux routine may in turn invoke the label map get routine **104** to identify a sequence of conversion routines for processing the packet. In this example, the label map get routine identifies the first three conversion routines, and the demux routine creates the member path entries **151**, **152**, **153** of the path for these conversion routines. Each path entry identifies a session for a conversion routine, and the sequence of path entries **151-155** identifies a path. The message send routine then queues the packet on the queue **149** for the path that is to process the packets of the message. The path thread **105** for the path retrieves the packet from the queue and invokes the message send routine **106** passing the packet and an indication of the path. The message send routine determines that the next session and conversion routine as indicated by path entry **151** has already been found. The message send routine then invokes the instance of the conversion routine for the session. The conversion routine processes the packet and then invokes the message send routine **107**. This processing continues until the message send routine invokes the demux routine **110** after the packet is processed by the conversion routine represented by path entry **153**. The demux routine examines the path and determines that it has no more path entries. The demux routine then invokes the label map get routine **111** to identify the conversion routines for further processing of the packet. When the conversion routines are identified, the demux routine adds path entries **154**, **155** to the path. The messages send routine invokes the conversion routine associated with path entry **154**. Eventually, the conversion routine associated with path entry **155** performs the final processing for the path.

The label map get routine identifies a sequence of "edges" for converting data in one format into another format. Each edge corresponds to a conversion routine for converting data from one format to another. Each edge is part of a "protocol" (or more generally a component) that may include multiple related edges. For example, a protocol may have edges that each convert data in one format into several different formats. Each edge has an input format and an output format. The label map get routine identifies a sequence of edges such that the output format of each edge is compatible with the input format of another edge in the sequence, except for the input format of the first edge in the sequence and the output format of the last edge in the sequence. FIG. **2** is a block diagram illustrating a sequence of edges. Protocol PI includes an edge for converting format D1 to format D2 and an edge for converting format D1 to format D3; protocol P2 includes an edge for converting format D2 to format D5, and so on. A 30 sequence for converting format D 1 to format D 15 is shown by the curved lines and is defined by the address "P 1: I, P2: 1, P3:2, P4:7." When a packet of data in format D I is processed by this sequence, it is converted to format DIS. During the process, the packet of data is sequentially converted to format D2, D5, and D13. The output format of protocol P2, edge 1 (i.e., P2: 1) is format D5, but the input format of P3:2 is format D10. The label map get routine uses an aliasing mechanism by which two formats, such as D5 and D10 are identified as being compatible. The use of aliasing allows different names of the same format or compatible formats to be correlated.

FIG. **3** is a block diagram illustrating components of the conversion system in one embodiment. The conversion system **300** can operate on a computer system with a central processing unit **301**, I/O devices **302**, and memory **303**. The **110** devices may include an Internet connection, a connection to various output devices such as a television, and a connection to various input devices such as a television receiver. The media mapping system may be stored as instructions on a computer-readable medium, such as a disk drive, memory, or data transmission medium. The data structures of the media mapping system may also be stored on a computer-readable medium. The conversion system includes drivers **304**, a• forwarding component **305**, a demux component **306**, a label map get component **307**, path data structures **308**, conversion routines **309**, and instance data **310**. Each driver receives data in a source format and forwards the data to the forwarding component. The forwarding component identifies the next conversion routine in the path and invokes that conversion routine to process a packet. The forwarding component may invoke the demux component to search for the next conversion routine and add that conversion routine to the path. The demux component may invoke the label map get component to identify the next conversion routine to process the packet. The demux component stores information defining the paths in the path structures. The conversion routines store their state information in the instance data.

FIG. **4** is a block diagram illustrating example path data structures in one embodiment. The demux component identifies a sequence of "edges" for converting data in one format into another format by invoking the label map get component. Each edge corresponds to a conversion routine for converting data from one format to another. As discussed above, each edge is part of a "protocol" that may include multiple related edges. For example, a protocol may have edges that each convert data in one format into several different formats. Each edge has as an input format ("input label") and an output format ("output label"). Each rectangle represents a session **410**, **420**, **430**, **440**, **450** for a protocol. A session corresponds to an instance of a protocol. That is, the session includes the protocol and state information associated with that instance of the protocol. Session **410** corresponds to a session for an Ethernet protocol; session **420** corresponds to a session for an IP protocol; and sessions **430**, **440**, **450** correspond to sessions for a TCP protocol. FIG. **4** illustrates three paths **461**, **462**, **463**. Each path includes edges **411**, **421**, **431**. The paths share the same Ethernet session **410** and IP session **420**, but each path has a unique TCP session **430**, **440**, **450**. Thus, path **461** includes sessions **410**, **420**, and **430**; path **462** includes sessions **410**, **420**, and **440**; and path **463** includes sessions **410**, **420**, and **450**. The conversion system represents each path by a

sequence of path entry structures. Each path entry structure is represented by a triangle. Thus, path **461** is represented by path entries **415**, **425**, and **433**. The conversion system represents the path entries of a path by a stack list. Each path also has a queue **471**, **472**, **473** associated with it. Each queue stores the messages that are to be processed by the conversion routines of the edges of the path. Each session includes a binding **412**, **422**, **432**, **442**, **452** that is represented by an oblong shape adjacent to the corresponding edge. A binding for an edge of a session represents those paths that include the edge. The binding **412** indicates that three paths are bound (or "nailed") to edge **411** of the Ethernet session **410**. The conversion system uses a path list to track the paths that are bound to a binding. The path list of binding **412** identifies path entries **413**, **414**, and **415**.

FIG. **5** is a block diagram that illustrates the interrelationship of the data structures of a path. Each path has a corresponding path structure **501** that contains status information and pointers to a message queue structure **502**, a stack list structure **503**, and a path address structure **504**. The status of a path can be extend, continue, or end. Each message handler returns a status for the path. The status of extend means that additional path entries should be added to the path. The status of end means that this path should end at this point and subsequent processing should continue at a new path. The status of continue means that the protocol does not care how the path is handled. In one embodiment, when a path has a status of continue, the system creates a copy of the path and extends the copy. The message queue structure identifies the messages (or packets of a message) that are queued up for processing by the path and identifies the path entry at where the processing should start. The stack list structure contains a list of pointers to the path entry structures **505** that comprise the path. Each path entry structure contains a pointer to the corresponding path data structure, a pointer to a map structure **507**, a pointer to a multiplex list **508**, a pointer to the corresponding path address structure, and a pointer to a member structure **509**. A map structure identifies the output label of the edge of the path entry and optionally a target label and a target key. A target key identifies the session associated with the protocol that converts the packet to the target label. (The terms "media," "label," and "format" are used interchangeably to refer to the output of a protocol.) The multiplex list is used during the demux process to track possible next edges when a path is being identified as having more than one next edge. The member structure indicates that the path entry represents an edge of a path and contains a pointer to a binding structure to which the path entry is associated (or "nailed"), a stack list entry is the position of the path entry within the associated stack list, a path list entry is the position of the path entry within the associated path list of a binding and an address entry is the position of the binding within the associated path address. A path address of a path identifies the bindings to which the path entries are bound. The path address structure contains a URL for the path, the name of the path identified by the address, a pointer to a binding list structure **506**, and the identification of the current binding within the binding list. The URL (e.g., "protocol://tcp(0)/ip (0)/eth(0)") identifies conversion routines (e.g., protocols and edges) of a path in a human-readable format. The URL (universal resource locator) includes a type field (e.g., "protocol") followed by a sequence of items (e.g., "tcp(0)"). The type field specifies the format of the following information in the URL, that specifies that the type field is followed by a sequence of items. Each item identifies a protocol and an edge (e.g., the protocol is "tcp" and the edge is "0"). In one embodiment, the items of a URL may also contain an identifier of state information that is to be used when processing a message. These URLs can be used to illustrate to a user various paths that are available for processing a message. The current binding is the last binding in the path as the path is being built. The binding list structure contains a list of pointers to the binding structures associated with the path. Each binding structure **510** contains a pointer to a session structure, a pointer to an edge structure, a key, a path list structure, and a list of active paths through the binding. The key identifies the state information for a session of a protocol. A path list structure contains pointers to the path entry structures associated with the binding.

FIG. **6** is a block diagram that illustrates the interrelationship of the data structures associated with a session. A session structure **601** contains the context for the session, a pointer to a protocol structure for the session, a pointer to a binding table structure **602** for the bindings associated with the session, and the key. The binding table structure contains a list of pointers to the binding structures **510** for the session. The binding structure is described above with reference to FIG. **5**. The path list structure **603** of the binding structure contains a list of pointers to path entry structures **505**. The path entry structures are described with reference to FIG. **5**.

FIGS. 7A, 7B, and 7C comprise a flow diagram illustrating the processing of the message send routine. The message send routine is passed a message along with the path entry associated with the session that last processed the message. The message send routine invokes the message handler of the next edge in the path or queues the message for processing by a path. The message handler invokes the demux routine to identify the next path entry of the path. When a driver receives a message, it invokes the message send routine passing a reference path entry. The message send routine examines the passed path entry to determine (1) whether multiple paths branch from the path of the passed path entry, (2) whether the passed path entry is a reference with an associated path, or (3) whether the passed path entry is a member with a next path entry. If multiple paths branch from the path of the passed path entry, then the routine recursively invokes the message send routine for each path. If the path entry is a reference with an associated path, then the driver previously invoked the message send routine, which associated a path with the reference path entry, and the routine places the message on the queue for the path. If the passed path entry is a member with a next path entry, then the routine invokes the message handler (i.e., conversion routine of the edge) associated with the next path entry. If the passed path entry is a reference without an associated path or is a member without a next path entry, then the routine invokes the demux routine to identify the next path entry. The routine then recursively invokes the messages send routine passing that next path entry. In decision block **701**, if the passed path entry has a multiplex list, then the path branches off into multiple paths and the routine continues at block **709**, else the routine continues at block **702**. A packet may be processed by several different paths. For example, if a certain message is directed to two different output devices, then the message is processed by two different paths. Also, a message may need to be processed by multiple partial paths when searching for a complete path. In decision block **702**, if the passed path entry is a member, then either the next path entry indicates a nailed binding or the path needs to be extended and the routine continues at block **704**, else the routine continues at block **703**. A nailed binding is a binding (e.g., edge and protocol) is associated with a session. In decision block **703**, the passed path entry

is a reference and if the passed path entry has an associated path, then the routine can queue the message for the associated path and the routine continues at block **703**A, else the routine needs to identify a path and the routine continues at block **707**. In block **703**A, the routine sets the entry to the first path entry in the path and continues at block **717**. In block **704**, the routine sets the variable position to the stack list entry of the passed path entry. In decision block **705**, the routine sets the variable next entry to the next path entry in the path. If there is a next entry in the path, then the next session and edge of the protocol have been identified and the routine continues at block **706**, else the routine continues at block **707**. In block **706**, the routine passes the message to the message handler of the edge associated with the next entry and then returns. In block **706**, the routine invokes the demux routine passing the passed message, the address of the passed path entry, and the passed path entry. The demux routine returns a list of candidate paths for processing of the message. In decision block **708**, if at least one candidate path is returned, then the routine continues at block **709**, else the routine returns.

Blocks **709**-**716** illustrate the processing of a list of candidate paths that extend from the passed path entry. In blocks **710**-**716**, the routine loops selecting each candidate path and sending the message to be process by each candidate path. In block **710**, the routine sets the next entry to the first path entry of the next candidate path. In decision block **711**, if all the candidate paths have not yet been processed, then the routine continues at block **712**, else the routine returns. In decision block **712**, if the next entry is equal to the passed path entry, then the path is to be extended and the routine continues at block **705**, else the routine continues at block **713**. The candidate paths include a first path entry that is a reference path entry for new paths or that is the last path entry of a path being extended. In decision block **713**, if the number of candidate paths is greater than one, then the routine continues at block **714**, else the routine continues at block **718**. In decision block **714**, if the passed path entry has a multiplex list associated with it, then the routine continues at block **716**, else the routine continues at block **715**. In block **715**, **11** the routine associates the list of candidate path with the multiplex list of the passed path entry and continues at block **716**. In block **716**, the routine sends the message to the next entry by recursively invoking the message send routine. The routine then loops to block **710** to select the next entry associated with the next candidate path.

Blocks **717**-**718** are performed when the passed path entry is a reference path entry that has a path associated with it. In block **717**, if there is a path associated with the next entry, then the routine continues at block **718**, else the routine returns. In block **718**, the routine queues the message for the path of the next entry and then returns.

FIG. **8** is a flow diagram of the demux routine. This routine is passed the packet (message) that is received, an address structure, and a path entry structure. The demux routine extends a path, creating one if necessary. The routine loops identifying the next binding (edge and protocol) that is to process the message and "nailing" the binding to a session for the message, if not already nailed. After identifying the nailed binding, the routine searches for the shortest path through the nailed binding, creating a path if none exists. In block **801**, the routine invokes the initialize demux routine. In blocks **802**-**810**, the routine loops identifying a path or portion of a path for processing the passed message. In decision block **802**, if there is a current status, which was returned by the demuxkey routine that was last invoked (e.g., continue, extend, end, or postpone), then the routine continues at block **803**, else the routine continues at block **811**. In block **803**, the routine invokes the get next binding routine. The get next binding routine returns the next binding in the path. The binding is the edge of a protocol. That routine extends the path as appropriate to include the binding. The routine returns a return status of break, binding, or multiple. The return status of binding indicates that the next binding in the path was found by extending the path as appropriate and the routine continues to "nail" the binding to a session as appropriate. The return status of multiple means that multiple trails (e.g., candidate paths) were identified as possible extensions of the path. In a decision block **804**, if the return status is break, then the routine continues at block **811**. If the return status is multiple, then the routine returns. If the return status is binding, then the routine continues at block **805**. In decision block **805**, if the retrieved binding is nailed as indicated by being assigned to a session, then the routine loops to block **802**, else the routine continues at block **806**. In block **806**, the routine invokes the get key routine of the edge associated with the binding. The get key routine creates the key for the session associated with the message. If a key cannot be created until subsequent bindings are processed or because the current binding is to be removed, then the get key routine returns a next binding status, else it returns a continue status. In decision block **807**, if the return status of the get key routine is next binding, then the routine loops to block **802** to get the next binding, else the routine continues at block **808**. In block **808**, the routine invokes the routine get session. The routine get session returns the session associated with the key, creating a new session if necessary. In block **809**, the routine invokes the routine nail binding. The routine nail binding retrieves the binding if one is already nailed to the session. Otherwise, that routine nails the binding to the session. In decision block **810**, if the nail binding routine returns a status of simplex, then the routine continues at block **811** because only one path can use the session, else the routine loops to block **802**. Immediately upon return from the nail binding routine, the routine may invoke a set map routine of the edge passing the session and a map to allow the edge to set its map. In block **811**, the routine invokes the find path routine, which finds the shortest path through the binding list and creates a path if necessary. In block **812**, the routine invokes the process path hopping routine, which determines whether the identified path is part of a different path. Path hopping occurs when, for example, IP fragments are built up along separate paths, but once the fragments are built up they can be processed by the same subsequent path.

FIG. **9** is a flow diagram of the initialize demux routine. This routine is invoked to initialize the local data structures that are used in the demux process and to identify the initial binding. The demux routine finds the shortest path from the initial binding to the final binding. If the current status is demux extend, then the routine is to extend the path of the passed path entry by adding additional path entries. If the current status is demux end, then the demux routine is ending the current path. If the current status is demux continue, then the demux routine is in the process of continuing to extend or in the process of starting a path identified by the passed address. In block **901**, the routine sets the local map structure to the map structure in the passed path entry structure. The map structure identifies the output label, the target label, and the target key. In the block **902**, the routine initializes the local message structure to the passed message structure and initializes the pointers path and address element to null. In block **903**, the routine sets of the variable saved status to **0** and the variable status to demux continue. The variable saved status is used to track the status of the demux process when backtracking to nail a binding whose nail was postponed. In decision block **904**, if the passed path entry is associated with a path, then the routine continues at block **905**, else the routine continues at block **906**. In block **905**, the routine sets the variable status to the status of that path. In block **906**, if the variable status is demux continue, then the routine continues at block **907**. If the variable status is demux end, then the routine continues at block **908**. If the variable status is demux extend, then the routine continues at block **909**. In block **907**, the status is demux continue, and the routine sets the local pointer path address to the passed address and continues at block **911**. In block **908**, the status is demux end, and the routine invokes the init end routine and continues at block **911**. In block **909**, the status is demux extend, and the routine sets the local path address to the address of the path that contains the passed path entry. In block **910**, the routine sets the address element and the current binding of the path address pointed to by the local pointer path address to the address entry of the member structure of the passed path entry. In the block **911**, the routine sets the local variable status to demux continue and sets the local binding list structure to the binding list structure from the local path address structure. In block **912**, the routine sets the local pointer current binding to the address of the current binding pointed to by local pointer path address and sets the local variable postpone to 0. In block **913**, the routine sets the function traverse to the function that retrieves the next data in a list and sets the local pointer session to null. The routine then returns.

FIG. **10** is a flow diagram of the init end routine. If the path is simplex, then the routine creates a new path from where the other one ended, else the routine creates a copy of the path. In block **1001**, if the binding of the passed path entry is simplex (i.e., only one path can be bound to this binding), then the routine continues at block **1002**, else the routine continues at block **1003**. In block **1002**, the routine sets the local pointer path address to point to an address structure that is a copy of the address structure associated with the passed path entry structure with its current binding to the address entry associated with the passed path entry structure, and then returns. In block **1003**, the routine sets the local pointer path address to point to an address structure that contains the URL of the path that contains the passed path entry. In block **1004**, the routine sets the local pointer element to null to initialize the selection of the bindings. In blocks **1005** through **1007**, the routine loops adding all the bindings for the address of the passed path entry that include and are before the passed path entry to the address pointed to by the local path address. In block **1005**, the routine retrieves the next binding from the binding list starting with the first. If there is no such binding, then the routine returns, else the routine continues at block **1006**. In block **1006**, the routine adds the binding to the binding list of the local path address structure and sets the current binding of the local variable path address. In the block **1007**, if the local pointer element is equal to the address entry of the passed path entry, then the routine returns, else the routine loops to block **1005** to select the next binding.

FIG. **11** is a flow diagram of a routine to get the next binding. This routine returns the next binding from the local binding list. If there is no next binding, then the routine invokes the routine label map get to identify the list of edges ("trails") that will map the output label to the target label. If only one trail is identified, then the binding list of path address is extended by the edges of the trail. If multiple trails are identified, then a path is created for each trail and the routine returns so that the demux process can be invoked for each created path. In block **1101**, the routine sets the local pointer binding to point to the next or previous (as indicated by the traverse function) binding in the local binding list. In block **1102**, if a binding was found, then the routine returns an indication that a binding was found, else the routine continues at block **1103**. In block **1103**, the routine invokes the label map get function passing the output label and target label of the local map structure. The label map get function returns a trail list. A trail is a list of edges from the output label to the target label. In decision block **1104**, if the size of the trail list is one, then the routine continues at block **1105**, else the routine continues at block **1112**. In blocks **1105-1111**, the routine extends the binding list by adding a binding data structure for each edge in the trail. The routine then sets the local binding to the last binding in the binding list. In block **1108**, the routine sets the local pointer current binding to point to the last binding in the local binding list. In block **1106**, the routine sets the local variable temp trail to the trail in the trail list. In block **1107**, the routine extends the binding list by temp trail by adding a binding for each edge in the trail. These bindings are not yet nailed. In block **1108**, the routine sets the local binding to point to the last binding in the local binding list. In decision block **1109**, if the local binding does not have a key for a session and the local map has a target key for a session, then the routine sets the key for the binding to the target key of the local map and continues at block **1110**, else the routine loops to block **1101** to retrieve the next binding in path. In block **1110**, the routine sets the key of the local binding to the target key of the local map. In block **1111**, the routine sets the target key of the local map to null and then loop to block **1101** to return the next binding. In decision block **1112**, if the local session is set, then the demultiplexing is already in progress and the routine returns a break status. In block **1113**, the routine invokes a prepare multicast paths routine to prepare a path entry for each trail in the trail list. The routine then returns a multiple status.

FIG. **12** is a flow diagram of the get key routine. The get key routine invokes an edge's demuxkey routine to retrieve a key for the session associated with the message. The key identifies the session of a protocol. The demux key routine creates the appropriate key for the message. The demux key routine returns a status of remove, postpone, or other. The status of remove indicates that the current binding should be removed from the path. The status of postpone indicates that the demux key routine cannot create the key because it needs information provided by subsequent protocols in the path. For example, a TCP session is defined by a combination of a remote and local port address and an IP address. Thus, the TCP protocol postpones the creating of a key until the IP protocol identifies the IP address. The get key routine returns a next binding status to continue at the next binding in the path. Otherwise, the routine returns a continue status. In block **1201**, the routine sets the local edge to the edge of the local binding (current binding) and sets the local protocol to the protocol of the local edge. In block **1202**, the routine invokes the demux key routine of the local edge passing the local message, local path address, and local map. The demux key routine sets the key in the local binding. In decision block **1203**, if the demux key routine returns a status of remove, then the routine continues at block **1204**. If the demux key routine returns a status of postpone, then the routine continues at block **1205**, else the routine continues at block **1206**. In block **1204**, the routine sets the flag of the local binding to indicate that the binding is to be removed and continues at block **1206**. In block **1205**, the routine sets the variable traverse to the function to list the next data, increments the variable postpone, and then returns a next binding status. In blocks **1206-1214**, the routine processes the postponing of the creating of a key. In blocks **1207-1210**, if the creating of a key has been postponed, then the routine indicates to backtrack on the path, save the demux status, and set the demux status to demux continue. In blocks **1211-1213**, if the creating of a key has not been postponed, then the routine indicates to continue forward in the path and to restore any saved demux status. The save demux status is the status associated by the binding where the backtrack started. In decision block **1206**, if the variable postpone is set, then the routine continues at block **1207**, else the routine continues at block **1211**. In block **1207**, the routine decrements the variable postpone and sets the variable traverse to the list previous data function. In decision block **1208**, if the variable saved status is set, then the routine continues at block **1210**, else the routine continues at block **1209**. The variable saved status contains the status of the demux process when the demux process started to backtrack. In block **1209**, the routine sets the variable saved status to the variable status. In block **1210**, the routine sets the variable status to demux continue and continues at block **1214**. In block **1211**, the routine sets the variable traverse to the list next data function. In decision block **1212**, if the variable saved status in set, then the routine continues at block **1213**, else the routine continues at block **1214**. In block **1213**, the routine sets the variable status to the variable saved status and sets the variable saved status to 0. In decision block **1214**, if the local binding indicates that it is to be removed, then the routine returns a next binding status, else the routine returns a continue status.

FIG. **13** is a flow diagram of the get session routine. This routine retrieves the session data structure, creating a data structure session if necessary, for the key indicated by the binding. In block **1301**, the routine retrieves the session from the session table of the local protocol indicated by the key of the local binding. Each protocol maintains a mapping from each key to the session associated with the key. In decision block **1302**, if there is no session, then the routine continues at block **1303**, else the routine returns. In block **1303**, the routine creates a session for the local protocol. In block **1304**, the routine initializes the key for the local session based on the key of the local binding. In block **1305**, the routine puts the session into the session table of the local protocol. In block **1306**, the routine invokes the create session function of the protocol to allow the protocol to initialize its context and then returns.

FIG. **14** is a flow diagram of the nail binding routine. This routine determines whether a binding is already associated with ("nailed to") the session. If so, the routine returns that binding. If not, the routine associates the binding with the session. The routine returns a status of simplex to indicate that only one path can extend through the nailed binding. In decision block **1401**, if the binding table of the session contains an entry for the edge, then the routine continues at block **1402**, else the routine continues at block **1405**. In block **1402**, the routine sets the binding to the entry from the binding table of the local session for the edge. In block **1403**, the routine sets the current binding to point to the binding from the session. In block **1404**, if the binding is simplex, then the routine returns a simplex status, else the routine returns. Blocks **1405** through **1410** are performed when there is no binding in the session for the edge. In block **1405**, the routine sets the session of the binding to the variable session. In block **1406**, the routine sets the key of the binding to the key from the session. In block **1407**, the routine sets the entry for the edge in the binding table of the local session to the binding. In block **1408**, the routine invokes the create binding function of the edge of the binding passing the binding so the edge can initialize the binding. If that function returns a status of remove, the routine continues at block **1409**. In block **1409**, the routine sets the binding to be removed and then returns.

FIG. **15** is a flow diagram of the find path routine. The find path routine identifies the shortest path through the binding list. If no such path exists, then the routine extends a path to include the binding list. In decision block **1501**, if the binding is simplex and a path already goes through this binding (returned as an entry), then the routine continues at block **1502**, else the routine continues at block **1503**. In block **1502**, the routine sets the path to the path of the entry and returns. In block **1503**, the routine initializes the pointers element and short entry to null. In block **1504**, the routine sets the path to the path of the passed path entry. If the local path is not null and its status is demux extend, then the routine continues at block **1509**, else the routine continues at block **1505**. In blocks **1505-1508**, the routine loops identifying the shortest path through the bindings in the binding list. The routine loops selecting each path through the binding. The selected path is eligible if it starts at the first binding in the binding list and the path ends at the binding. The routine loops setting the short entry to the shortest eligible path found so far. In block **1505**, the routine sets the variable first binding to the first binding in the binding list of the path address. In block **1506**, the routine selects the next path (entry) in the path list of the binding starting with the first. If a path is selected (indicating that there are more paths in the binding), then the routine continues at block **1507**, else the routine continues at block **1509**. In block **1507**, the routine determines whether the selected path starts at the first binding in the binding list, whether the selected path ends at the last binding in the binding list, and whether the number of path entries in the selected path is less than the number of path entries in the shortest path selected so far. If these conditions are all satisfied, then the routine continues at block **1508**, else the routine loops to block **1506** to select the next path (entry). In block **1508**, the routine sets the shortest path (short entry) to the selected path and loops to block **1506** to select the next path through the binding. In block **1509**, the routine sets the selected path (entry) to the shortest path. In decision block **1510**, if a path has been found, then the routine continues at block **1511**, else the routine continues at block **1512**. In block **1511**, the routine sets the path to the path of the selected path entry and returns. Blocks **1512-1516** are performed when no paths have been found. In block **1512**, the routine sets the path to the path of the passed path entry. If the passed path entry has a path and its status is demux extend, then the routine continues at block **1515**, else the routine continues at block **1513**. In block **1513**, the routine creates a path for the path address. In block **1514**, the routine sets the variable element to null and sets the path entry to the first element in the stack list of the path. In block **1515**, the routine sets the variable element to be address entry of the member of the passed path entry and sets the path entry to the passed path entry. In block **1516**, the routine invokes the extend path routine to extend the path and then returns. The extend path routine creates a path through the bindings of the binding list and sets the path status to the current demux status.

FIG. **16** is a flow diagram of the process of path hopping routine. Path hopping occurs when the path through the

binding list is not the same path as that of the passed path entry. In decision block **1601**, if the path of the passed path entry is set, then the routine continues at block **1602**, else the routine continues at block **1609**. In decision block **1602**, if the path of the passed path entry is equal to the local path, then the routine continues at **1612**, else path hopping is occurring and the routine continues at block **1603**. In blocks **1603-1607**, the routine loops positioning pointers at the first path entries of the paths that are not at the same binding. In block **1603**, the routine sets the variable old stack to the stack list of the path of the passed path entry. In block **1604**, the routine sets the variable new stack to the stack list of the local path. In block **1605**, the routine sets the variable old element to the next element in the old stack. In block **1606**, the routine sets the variable element to the next element in the new stack. In decision block **1607**, the routine loops until the path entry that is not in the same binding is located. In decision block **1608**, if the variable old entry is set, then the routine is not at the end of the hopped from path and the routine continues at block **1609**, else routine continues at block **1612**. In block **1609**, the routine sets the variable entry to the previous entry in the hopped-to path. In block **1610**, the routine sets the path of the passed path entry to the local path. In block **1611**, the routine sets the local entry to the first path entry of the stack list of the local path. In block **1612**, the routine inserts an entry into return list and then returns.

Although the conversion system has been described in terms of various embodiments, the invention is not limited to these embodiments. Modification within the spirit of the invention will be apparent to those skilled in the art. For example, a conversion routine may be used for routing a message and may perform no conversion of the message. Also, a reference to a single copy of the message can be passed to each conversion routine or demuxkey routine. These routines can advance the reference past the header information for the protocol so that the reference is positioned at the next header. After the demux process, the reference can be reset to point to the first header for processing by the conversion routines in sequence. The scope of the invention is defined by the claims that follow.

What is claimed is:

1. A method, comprising:

receiving, at a computing device having a processing circuit, a packet of a message;

determining, by the computing device, a key value for the packet, wherein the key value is determined based on one or more headers in the packet; and

using, by the computing device, the key value to determine whether the computing device is currently storing a previously created path for the key value

in response to determining that no path is currently stored for the key value, the computing device:

identifying, using the key value, one or more routines for processing the packet, including a routine that is used to execute a Transmission Control Protocol (TCP) to convert packets having a TCP format into a different format;

creating a path using the identified one or more routines wherein the created path stores state information for at least one of the identified one or more routines and specifies an ordering in which the identified one or more routines are to be performed to process the packet; and

processing the packet using the created path.

* * * * *

EXHIBIT 9

US010225378B2

(12) **United States Patent**
Balassanian

(10) **Patent No.: US 10,225,378 B2**
(45) **Date of Patent: *Mar. 5, 2019**

(54) METHOD AND SYSTEM FOR DATA DEMULTIPLEXING

(71) Applicant: **Implicit, LLC**, Seattle, WA (US)

(72) Inventor: **Edward Balassanian**, Austin, TX (US)

(73) Assignee: **Implicit, LLC**, Seattle, WA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **16/043,069**

(22) Filed: **Jul. 23, 2018**

(65) **Prior Publication Data**

US 2018/0332145 A1 Nov. 15, 2018

Related U.S. Application Data

(63) Continuation of application No. 15/450,790, filed on Mar. 6, 2017, now Pat. No. 10,033,839, which is a
(Continued)

(51) **Int. Cl.**
H04L 12/58 (2006.01)
H04L 29/06 (2006.01)
(Continued)

(52) **U.S. Cl.**
CPC ***H04L 69/08*** (2013.01); ***H04L 29/06*** (2013.01); ***H04L 45/00*** (2013.01);
(Continued)

(58) **Field of Classification Search**
None
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,298,674	A	3/1994	Yun
5,392,390	A	2/1995	Crozier

(Continued)

FOREIGN PATENT DOCUMENTS

EP	0408132	1/1991
EP	0807347	11/1997

(Continued)

OTHER PUBLICATIONS

Michael Baentsch, et al., "WebMake: Integrating distributed software development in a structure-enhanced Web," Computer Networks and ISDN Systems 27 (1995), pp. 789-800.

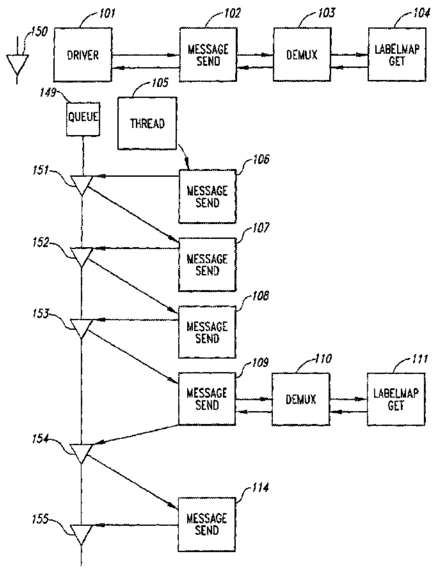
(Continued)

Primary Examiner — Duc T Duong
(74) *Attorney, Agent, or Firm* — Meyertons, Hood, Kivlin, Kowert & Goetzel, P.C.

(57) **ABSTRACT**

A method and system for demultiplexing packets of a message is provided. The demultiplexing system receives packets of a message, identifies a sequence of message handlers for processing the message, identifies state information associated with the message for each message handler, and invokes the message handlers passing the message and the associated state information. The system identifies the message handlers based on the initial data type of the message and a target data type. The identified message handlers effect the conversion of the data to the target data type through various intermediate data types.

20 Claims, 16 Drawing Sheets



Related U.S. Application Data

continuation of application No. 15/050,027, filed on Feb. 22, 2016, now Pat. No. 9,591,104, which is a continuation of application No. 14/230,952, filed on Mar. 31, 2014, now Pat. No. 9,270,790, which is a continuation of application No. 13/911,324, filed on Jun. 6, 2013, now Pat. No. 8,694,683, which is a continuation of application No. 13/236,090, filed on Sep. 19, 2011, now abandoned, which is a continuation of application No. 10/636,314, filed on Aug. 6, 2003, now Pat. No. 8,055,786, which is a continuation of application No. 09/474,664, filed on Dec. 29, 1999, now Pat. No. 6,629,163.

(51) **Int. Cl.**

H04L 29/08	(2006.01)
H04L 29/12	(2006.01)
H04L 12/701	(2013.01)

(52) **U.S. Cl.**
CPC ***H04L 61/2007*** (2013.01); ***H04L 61/6063*** (2013.01); ***H04L 67/02*** (2013.01); ***H04L 69/16*** (2013.01); ***H04L 69/18*** (2013.01); ***H04L 69/22*** (2013.01); ***H04L 69/32*** (2013.01)

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,414,833	A	5/1995	Hershey et al.
5,425,029	A	6/1995	Hluchyj et al.
5,568,478	A	10/1996	van Loo, Jr. et al.
5,627,997	A	5/1997	Pearson et al.
5,710,917	A	1/1998	Musa et al.
5,727,159	A	3/1998	Kikinis
5,740,430	A	4/1998	Rosenberg et al.
5,761,651	A	6/1998	Hasebe
5,768,521	A	6/1998	Dedrick
5,826,027	A	10/1998	Pedersen et al.
5,835,726	A	11/1998	Shwed et al.
5,842,040	A	11/1998	Hughes et al.
5,848,233	A	12/1998	Radia et al.
5,848,246	A	12/1998	Gish
5,848,415	A	12/1998	Guck
5,854,899	A	12/1998	Callon et al.
5,870,479	A	2/1999	Feiken et al.
5,896,383	A	4/1999	Wakeland
5,898,830	A	4/1999	Wesinger, Jr. et al.
5,918,013	A	6/1999	Mighdoll et al.
5,983,348	A	11/1999	Ji
5,987,256	A	11/1999	Wu et al.
5,991,299	A	11/1999	Radogna et al.
5,991,806	A	11/1999	McHann, Jr.
6,032,150	A	2/2000	Nguyen
6,035,339	A	3/2000	Agraharam et al.
6,047,002	A	4/2000	Hartmann et al.
6,067,575	A	5/2000	McManis et al.
6,091,725	A	7/2000	Cheriton et al.
6,094,679	A	7/2000	Teng et al.
6,101,189	A	8/2000	Tsuruoka
6,101,320	A	8/2000	Schuetze et al.
6,104,500	A	8/2000	Alam et al.
6,104,704	A	8/2000	Buhler et al.
6,111,893	A	8/2000	Volftsun et al.
6,112,250	A	8/2000	Appelman
6,115,393	A	9/2000	Engel et al.
6,119,165	A	9/2000	Li et al.
6,119,236	A	9/2000	Shipley
6,122,666	A	9/2000	Beurket et al.
6,128,624	A	10/2000	Papiemiak et al.
6,130,917	A	10/2000	Monroe
6,141,749	A	10/2000	Coss et al.
6,151,390	A	11/2000	Volftsun et al.
6,157,622	A	12/2000	Tanaka et al.
6,167,441	A	12/2000	Himmel
6,192,419	B1	2/2001	Aditham et al.
6,199,054	B1	3/2001	Khan et al.
6,212,550	B1	4/2001	Segur
6,222,536	B1	4/2001	Kihl et al.
6,226,267	B1	5/2001	Spinney et al.
6,243,667	B1	6/2001	Kerr et al.
6,246,678	B1	6/2001	Erb et al.
6,259,781	B1	7/2001	Crouch et al.
6,275,507	B1	8/2001	Anderson et al.
6,278,532	B1	8/2001	Heimendinger et al.
6,292,827	B1	9/2001	Raz
6,356,529	B1	3/2002	Zarom
6,359,911	B1	3/2002	Movshovich et al.
6,374,305	B1	4/2002	Gupta et al.
6,401,132	B1	6/2002	Bellwood et al.
6,404,775	B1	6/2002	Leslie et al.
6,405,254	B1	6/2002	Hadland
6,426,943	B1	7/2002	Spinney et al.
6,493,348	B1	12/2002	Gelman et al.
6,504,843	B1	1/2003	Cremin et al.
6,519,636	B2	2/2003	Engel et al.
6,560,236	B1	5/2003	Varghese et al.
6,574,610	B1	6/2003	Clayton et al.
6,578,084	B1	6/2003	Moberg et al.
6,598,034	B1	7/2003	Kloth
6,629,163	B1	9/2003	Balassanian
6,650,632	B1	11/2003	Volftsun et al.
6,651,099	B1	11/2003	Dietz et al.
6,678,518	B2	1/2004	Eerola
6,680,922	B1	1/2004	Jorgensen
6,701,432	B1	3/2004	Deng et al.
6,711,166	B1	3/2004	Amir et al.
6,772,413	B2	8/2004	Kuznetsov
6,785,730	B1	8/2004	Taylor
6,865,735	B1	3/2005	Sirer et al.
6,871,179	B1	3/2005	Kist et al.
6,889,181	B2	5/2005	Kerr et al.
6,937,574	B1	8/2005	Delaney et al.
6,957,346	B1	10/2005	Kivinen et al.
6,959,439	B1	10/2005	Boike
7,233,569	B1	6/2007	Swallow
7,233,948	B1	6/2007	Shamoon et al.
7,281,036	B1	10/2007	Lu et al.
7,383,341	B1	6/2008	Saito et al.
7,443,858	B1 *	10/2008	Cheriton H04L 12/4608 370/395.1
7,711,857	B2	5/2010	Balassanian
8,055,786	B2	11/2011	Balassanian
8,694,683	B2	4/2014	Balassanian
2001/0037397	A1 *	11/2001	Boucher G06F 5/10 709/230
2002/0156927	A1 *	10/2002	Boucher H04L 29/06 709/250
2003/0142669	A1	7/2003	Kubota et al.
2004/0015609	A1	1/2004	Brown et al.
2004/0158793	A1 *	8/2004	Blightman H04L 29/06 714/758
2006/0209830	A1 *	9/2006	Oguchi H04L 45/00 370/758
2007/0067497	A1 *	3/2007	Craft H04L 67/1097 709/250
2008/0250045	A1	10/2008	Balassanian et al.
2009/0083763	A1	3/2009	Sareen et al.
2009/0265695	A1	10/2009	Karino
2009/0310485	A1	12/2009	Averi et al.
2015/0032691	A1	1/2015	Hall et al.

FOREIGN PATENT DOCUMENTS

EP	0817031	1/1998
JP	H10-55279	2/1998
JP	H1049354	2/1998
JP	H10-74153	3/1998

(56) **References Cited**

FOREIGN PATENT DOCUMENTS

JP	H10-289215	10/1998
WO	99/35799	7/1999

OTHER PUBLICATIONS

Dan Decasper, et al., "A Scalable, High Performance Active Network Node," Apr. 1998, 21 pages.
John J. Hartman, et al., "Joust: A Platform for Liquid Software," Computer, IEEE, 1999, pp. 50-56.
David Mosberger, et al., "Making Paths Explicit in the Scout Operating System," Proceedings of the USENIX 2nd Symposium on Operating Systems Design and Implementation, Oct. 1996, 16 pages.
Oliver Spatscheck, et al., "Escort: A Path-Based OS Security Architecture," TR 97-17, Nov. 26, 1997, 17 pages.
Dan Decasper, et al., "DAN: Distrubuted Code Caching for Active Networks," IEEE, 1998, pp. 609-616.
Non-Final Office Action in Inter Partes Reexamination Control No. 95/000,659 dated Aug. 16, 2013, 107 pages.
Decision on Petition in Reexamination Control No. 95/000,659 dated Aug. 19, 2013, 3 pages.
Response to Non-Final Office Action in Reexamination Control No. 95/000,659 dated Oct. 2, 2013 including Exhibits A-C, 37 pages.
Decision on Petition in Reexamination Control No. 95/000,660 dated Jul. 30, 2013, 12 pages.
Non-Final Office Action in Inter Partes Reexamination Control No. 95/000,660 dated Aug. 30, 2013, 23 pages.
RFC: 791. Internet Protocol: DARPA Internet Program Protocol Specification, Sep. 1981, prepared for Defense Advanced Research Projects Agency Information Processing Techniques Office by Information Sciences Institute University of Southern California, 52 pages.
2015 WL 2194627, United States District Court, N.D. California, *Implicit L.L.C.* , Plaintiff, v. *F5 Networks, Inc.*, Defendant, Case No. 14-cv-02856-SI, signed May 6, 2015, 14 pages.
Defendants' Invalidity Contentions Pursuant to Local Patent Rules 3-3 and 3-4, United States District Court Eastern District of Texas Tyler Division, *Implicit, LLC* v. *Trend Micro, Inc., Ericsson Inc., Huawei Technologies USA, Inc., NEC Corporation of America, Nokia Solutions and Networks US LLC*; Sep. 2, 2016, 53 pages.
Exhibits A-1-A16 Invalidity of U.S. Pat. No. 8,694,683, Defendants' Invalidity Contentions Pursuant to Local Patent Rules 3-3 and 3-4, United States District Court Eastern District of Texas Tyler Division, Sep. 2, 2016, 425 pages.
Exhibits B-1-B13 Invalidity of U.S. Pat. No. 9,270,790, Defendants' Invalidity Contentions Pursuant to Local Patent Rules 3-3 and 3-4, United States District Court Eastern District of Texas Tyler Division, Sep. 2, 2016, 369 pages.
Exhibits C-1-C21 Invalidity of U.S. Pat. No. 8,856,779, Defendants' Invalidity Contentions Pursuant to Local Patent Rules 3-3 and 3-4, United States District Court Eastern District of Texas Tyler Division, Sep. 2, 2016, 646 pages.
Exhibits D-1-D21 Invalidity of U.S. Pat. No. 9,325,740, Defendants' Invalidity Contentions Pursuant to Local Patent Rules 3-3 and 3-4, United States District Court Eastern District of Texas Tyler Division, dated Sep. 2, 2016, 419 pages.
Exhibits E-1-E20 Invalidity of U.S. Pat. No. 6,324,685, Defendants' Invalidity Contentions Pursuant to Local Patent Rules 3-3 and 3-4, United States District Court Eastern District of Texas Tyler Division, dated Sep. 2, 2016, 416 pages.
Alexander, D. et al., "The SwitchWare Active Network Architecture", Jun. 6, 1998, IEEE.
Antoniazzi, S. et al., "An Open Software Architecture for Multimedia Consumer Terminals", Central Research Labs, Italy; Alcatel SEL Research Centre, Germany, ECMAST 1997.
Arbanowski, Stefan, "Generic Description of Telecommunication Services and Dynamic Resource Selection in Intelligent Communication Environments", Thesis, Technische Universitat Berlin, Oct. 9, 1996 (3 documents).
Arbanowski, S., et al., Service Personalization for Unified Messaging Systems, Jul. 6-8, 1999, The Fourth IEEE Symposium on Computers and Communications, ISCC '99, Red Sea, Egypt.
Atkinson, R., "Security Architecture for the Internet Protocol", Aug. 1995, Naval Research Laboratory.
Atkinson, R., "IP Authentication Header", Aug. 1995, Naval Research Laboratory.
Atkinson, R., "IP Encapsulating Security Payload (ESP)", Aug. 1995, Naval Research Laboratory.
Back, G., et al., Java Operating Systems: Design and Implementation, Aug. 1998, Technical Report UUCS-98-015, University of Utah.
Baker, Dr. Sean, "CORBA Implementation Issues", 1994, IONA Technologies, O'Reilly Institute Dublin, Ireland.
Barrett, R., et al., "Intermediaries: New Places for Producing and Manipulating Web Content", 1998, IBM Almaden Research Center, Elsevier Science.
Bellare, M., et al., "A Concrete Security Treatment of Symmetric Encryption: Analysis of the DES Modes of Operation", Aug. 15, 1997, Dept. of Computer Science and Engineering, University of California, San Diego.
Bellare, M., et al., "A Concrete Security Treatment of Symmetric Encryption: Analysis of the DES Modes of Operation", Aug. 15, 1997, IEEE.
Bellare, M., et al., "XOR MACs: New Methods for Message Authentication Using Finite Pseudorandom Functions", 1995, CRYPTO '95, LNCS 963, pp. 15-28, Springer-Verlag Berlin Heidelberg.
Bellissard, L., et al., "Dynamic Reconfiguration of Agent-Based Applications", Third European Research Seminar on Advances in Distributed Systems, (ERSADS '99) Madeira Island.
Bolding, Darren, "Network Security, Filters and Firewalls", 1995, www.acm.org/crossroads/xrds2-1/security.html.
Booch, G., et al., "Software Engineering with ADA", 1994, Third Edition, The Benjamin/Cummings Publishing Company, Inc. (2 documents).
Breugst, et al., "Mobile Agents—Enabling Technology for Active Intelligent Network Implementation", May/Jun. 1998, IEEE Network.
"C Library Functions", AUTH(3) Sep. 17, 1993, Solbourne Computer, Inc.
Chapman, D., et al., "Building Internet Firewalls", Sep. 1995, O'Reilly & Associates, Inc.
CheckPoint FireWall-1 Technical White Paper, Jul. 18, 1994, CheckPoint Software Technologies, Ltd.
CheckPoint FireWall-1 White Paper, Sep. 1995, Version 2.0, CheckPoint Software Technologies, Ltd.
Command Line Interface Guide P/N 093-0011-000 Rev C Version 2.5, 2000-2001, NetScreen Technologies, Inc.
Coulson, G. et al., "A CORBA Compliant Real-Time Multimedia Platform for Broadband Networks", Lecture Notes in Computer Science, 1996, Trends in Distributed Systems CORBA and Beyond.
Cox, Brad, "SuperDistribution, Objects As Property on the Electronic Frontier", 1996, Addison-Wesley Publishing Company.
Cranes, et al., "A Configurable Protocol Architecture for CORBA Environments", Autonomous Decentralized Systems 1997 Proceedings ISADS, Third International Symposium Apr. 9-11, 1997.
Curran, K., et al., "CORBA Lacks Venom", University of Ulster, Northern Ireland, UK 2000.
Dannert, Andreas, "Call Logic Service for a Personal Communication Supporting System", Thesis, Jan. 20, 1998, Technische Universitat Berlin.
Darpa Internet Program Protocol Specification, "Transmission Control Protocol", Sep. 1981, Information Sciences Institute, California.
Darpa Internet Program Protocol Specification, "Internet Protocol", Sep. 1981, Information Sciences Institute, California.
Decasper, D., et al., "Crossbow: A Toolkit for Integrated Services over Cell Switched IPv6", 1997, Computer Engineering and Networks Laboratory, ETH Zurich, Switzerland.
Decasper, D., et al., "Router Plugins a Software Architecture for Next Generation Routers", 1998, Proceedings of ACM SIGCONM '98.
Deering, S., et al., Internet Protocol, Version 6 (IPv6) Specification, Dec. 1998, Nokia, The Internet Society.

(56) **References Cited**

OTHER PUBLICATIONS

Deering, S., et al., Internet Protocol, Version 6 (IPv6) Specification, Dec. 1995, Network Working Group, RFC 1883.
Dutton, et al, "Asynchronous Transfer Mode Technical Overview (ATM)", Second Edition; IBM, Oct. 1995, 2nd Edition, Prentice Hall PTR, USA.
Eckardt, T., et al., "Application of X.500 and X.700 Standards for Supporting Personal Communications in Distributed Computing Environments", 1995, IEEE.
Eckardt, T., et al., "Personal Communications Support based on TMN and TINA Concepts", 1996, IEEE Intelligent Network Workshop (IN '96), Apr. 21-24, Melbourne, Australia.
Eckardt, T., et al., "Beyond IN and UPT—A Personal Communications Support System Based on TMN Concepts", Sep. 1997, IEEE Journal on Selected Areas in Communications, vol. 15, No. 7.
Egevang, K., et al., "The IP Network Address Translator (NAT)", May 1994, Network Working Group, RFC 1631.
Estrin, D., et al., "Visa Protocols for Controlling Inter-Organizational Datagram Flow", Dec. 1998, Computer Science Department, University of Southern California and Digital Equipment Corporation.
Faupel, M., "Java Distribution and Deployment", Oct. 9, 1997, APM Ltd., United Kingdom.
Felber, P., "The CORBA Object Group Service: A Service Approach to Object Groups in CORBA", Thesis, 1998, Ecole Polytechnique Federale de Lausanne, Switzerland.
Fish, R., et al., "DRoPS: Kernel Support for Runtime Adaptable Protocols", Aug. 25-27, 1998, IEEE 24th Euromicro Conference, Sweden.
Fiuczynski, M., et al., "An Extensible Protocol Architecture for Application-Specific Networking", 1996, Department of Computer Science and Engineering, University of Washington.
Franz, Stefan, "Job and Stream Control in Heterogeneous Hardware and Software Architectures", Apr. 1998, Technische Universitat, Berlin (2 documents).
Fraser, T., "DTE Firewalls: Phase Two Measurement and Evaluation Report", Jul. 22, 1997, Trusted Information Systems, USA.
Gazis, V., et al., "A Survey of Dynamically Adaptable Protocol Stacks", first Quarter 2010, IEEE Communications Surveys & Tutorials, vol. 12, No. 1, 1st Quarter.
Gokhale, A., et al., "Evaluating the Performance of Demultiplexing Strategies for Real-Time CORBA", Nov. 1997, GLOBECOM.
Gokhale, A., et al., "Measuring and Optimizing CORBA Latency and Scalability Over High-Speed Networks", Apr. 1998, IEEE Transaction on Computers, vol. 47, No. 4; Proceedings of the International Conference on Distributed Computing Systems (ICDCS '97) May 27-30, 1997.
Gokhale, A., et al., "Operating System Support for High-Performance, Real-Time CORBA", 1996.
Gokhale, A., et al., "Principles for Optimizing CORBA Internet Inter-ORB Protocol Performance", Jan. 9, 1998, Proceedings of the HICSS Conference, Hawaii.
Gong, Li, "Java Security: Present and Near Future", May/Jun. 1997, IEEE Micro.
Gong, Li, "New Security Architectural Directions for Java (Extended Abstract)", Dec. 19, 1996, IEEE.
Gong, Li, "Secure Java Class Loading", Nov./Dec. 1998, IEEE Internet.
Goos, G., et al., "Lecture Notes in Computer Science: Mobile Agents and Security", 1998, Springer-Verlag Berlin Heidelberg.
Goralski, W., "Introduction to ATM Networking", 1995, McGraw-Hill Series on Computer Communications, USA.
Hamzeh, K., et al., Layer Two Tunneling Protocol "L2TP", Jan. 1998, PPP Working Group, Internet Draft.
Harrison, T., et al., "The Design and Performance of a Real-Time CORBA Event Service", Aug. 8, 1997,Proceedings of the OOPSLA '97 Conference, Atlanta, Georgia in Oct. 1997.
Huitema, Christian, "IPv6 The New Internet Protocol", 1997 Prentice Hall, Second Edition.
Hutchins, J., et al., "Enhanced Internet Firewall Design Using Stateful Filters Final Report", Aug. 1997, Sandia Report; Sandia National Laboratories.
IBM, Local Area Network Concepts and Products: Routers and Gateways, May 1996.
Juniper Networks Press Release, Juniper Networks Announces Junos, First Routing Operating System for High-Growth Internet Backbone Networks, Jul. 1, 1998, Juniper Networks.
Juniper Networks Press Release, Juniper Networks Ships the Industry's First Internet Backbone Router Delivering Unrivaled Scalability, Control and Performance, Sep. 16, 1998, Juniper Networks.
Karn, P., et al., "The ESP DES-CBC Transform", Aug. 1995, Network Working Group, RFC 1829.
Kelsey, J. et al., "Authenticating Outputs of Computer Software Using a Cryptographic Coprocessor", Sep. 1996, CARDIS.
Krieger, D., et al., "The Emergence of Distributed Component Platforms", Mar. 1998, IEEE.
Krupczak, B., et al., "Implementing Communication Protocols in Java", Oct. 1998, IEEE Communications Magazine.
Krupczak, B., et al., "Implementing Protocols in Java: The Price of Portability", 1998, IEEE.
Lawson, Stephen, "Cisco NetFlow Switching Speeds Traffic Routing", Jul. 7, 1997, Infoworld.
Li, S., et al., "Active Gateway: A Facility for Video Conferencing Traffic Control", Feb. 1, 1997, Purdue University; Purdue e-Pubs; Computer Science Technical Reports.
Magedanz, T., et al., "Intelligent Agents: An Emerging Technology for Next Generation Telecommunications?", 1996, IEEE.
Mills, H., et al., "Principles of Information Systems Analysis and Design", 1986, Academic Press, Inc. (2 documents).
Mosberger, David, "*Scout: A Path-Based Operating System*", Doctoral Dissertation Submitted to the University of Arizona, 1997 (3 documents).
Muhugusa, M., et al., "COMSCRIPT : An Environment for the Implementation of Protocol Stacks and their Dynamic Reconfiguration", Dec. 1994.
Nelson, M., et al., The Data Compression Book, 2nd Edition, 1996, M&T Books, A division of MIS Press, Inc.
NetRanger User's Guide, 1996, WheelGroup Corporation.
NetScreen Command Line Reference Guide, 2000, P/N 093-0000-001 Rev A, NetScreen Technologies, Inc., USA.
NetScreen Command Line Reference Guide, 2000, P/N 093-0000-001 NetScreen Technologies, Inc., USA.
NetScreen Concepts and Examples ScreenOS Reference Guide, 1998-2001, Version 2.5 P/N 093-0039-000 Rev. A, NetScreen Technologies, Inc.
NetScreen Products Webpage, wysiwyg://body_bottom.3/http://www...een.com/products/products.html 1998-1999, NetScreen Technologies, Inc.
NetScreen WebUI, Reference Guide, Version 2.5.0 P/N 093-0040-000 Rev. A, 2000-2001, NetScreen Technologies, Inc.
NetStalker Installation and User's Guide, 1996, Version 1.0.2, Haystack Labs, Inc.
Niculescu, Dragos, "Survey of Active Network Research", Jul. 14, 1999, Rutgers University.
Nortel Northern Telecom, "ISDN Primary Rate User-Network Interface Specification", Aug. 1998.
Nygren, Erik, "The Design and Implementation of a High-Performance Active Network Node", Thesis, Feb. 1998, MIT.
Osbourne, E., "Morningstar Technologies SecureConnect Dynamic Firewall Filter User's Guide", Jun. 14, 1995, V. 1.4, Morning Star Technologies, Inc.
Padovano, Michael, "Networking Applications on UNIX System V Release 4," 1993 Prentice Hall, USA (2 documents).
Pfeifer, T., "Automatic Conversion of Communication Media", 2000, GMD Research Series, Germany.
Pfeifer, T., "Automatic Conversion of Communication Media", Thesis, 1999, Technischen Universitat Berlin, Berlin.
Pfeifer, T., et al., "Applying Quality-of-Service Parametrization for Medium-to-Medium Conversion", Aug. 25-28, 1996, 8th IEEE Workshop on Local and Metropolitan Area Networks, Potsdam, Germany.

(56) **References Cited**

OTHER PUBLICATIONS

Pfeifer, T., "Micronet Machines—New Architectural Approaches for Multimedia End-Systems", 1993 Technical University of Berlin.
Pfeifer, T., "On the Convergence of Distributed Computing and Telecommunications in the Field of Personal Communications", 1995, KiVS, Berlin.
Pfeifer, T., "Speech Synthesis in the Intelligent Personal Communication Support System (IPCSS)", Nov. 2-3, 1995, 2nd 'Speak!' Workshop on Speech Generation in Multimodal Information Systems and Practical Applications.
Pfeifer, T., et al., "Generic Conversion of Communication Media for Supporting Personal Mobility", Nov. 25-27, 1996, Proc. of the Third COST 237 Workshop: Multimedia Telecommunications and Applications.
Pfeifer, T., et al., "Intelligent Handling of Communication Media", Oct. 29-31, 1997, 6th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS) Tunis.
Pfeifer, T., et al., "Resource Selection in Heterogeneous Communication Environments using the Teleservice Descriptor", Dec. 15-19, 1997, Proceedings from the 4th COST 237 Workshop: From Multimedia Services to Network Services, Lisboa.
Pfeifer, T., et al., Mobile Guide—Location-Aware Applications from the Lab to the Market, 1998, IDMS '98, LNCS 1483, pp. 15-28.
Pfeifer, T., et al., "The Active Store providing Quality Enhanced Unified Messaging", Oct. 20-22, 1998, 5th Conference on computer Communications, AFRICOM-CCDC '98, Tunis.
Pfeifer, T., et al., "*A Modular Location-Aware Service and Application Platform*", 1999, Technical University of Berlin.
Plagemann, T., et al., "*Evaluating Crucial Performance Issues of Protocol Configuration in DaCaPo*", 1994, University of Oslo.
Psounis, Konstantinos, "*Active Networks: Applications, Security Safety, and Architectures*", First Quarter 1999, IEEE Communications Surveys.
Rabiner, Lawrence, "*Applications of Speech Recognition in the Area of Telecommunications*", 1997, IEEE.
Raman, Suchitra, et al, "*A Model, Analysis, and Protocol Framework for Soft State-based Communications*", Department of EECS, University of California, Berkeley.
Rogaway, Phillip, "*Bucket Hashing and its Application to Fast Message Authentication*", Oct. 13, 1997, Department of Computer Science, University of California.
Schreier, B., et al., "*Remote Auditing of Software Outputs Using a Trusted CoProcessor*", 1997, Elsevier Paper Reprint 1999.
Tennenhouse, D., et al., "*From Internet to ActiveNet* ", Laboratory of Computer Science, MIT, 1996.
Tudor, P., "*Tutorial MPEG-2 Video Compression*", Dec. 1995, Electronics & Communication Engineering Journal.
US Copyright Webpage of Copyright Title, "*IPv6: the New Internet Protocol*", by Christian Huitema, 1998 Prentice Hall.
Van der Meer, et al., "*An Approach for a 4th Generation Messaging System*", Mar. 21-23, 1999, The Fourth International Symposium on Autonomous Decentralized Systems ISADS '99, Tokyo.
Van der Meer, Sven, "*Dynamic Configuration Management of the Equipment in Distributed Communication Environments*", Thesis, Oct. 6, 1996, Berlin (3 documents).
Van Renesse, R. et al., "*Building Adaptive Systems Using Ensemble*", Cornell University Jul. 1997.
Venkatesan, R., et al., "*Threat-Adaptive Security Policy*", 1997, IEEE.
Wetherall, D., et al., "*The Active IP Option*", Sep. 1996, Proceedings of the 7th ACM SIGOPS European Workshop, Connemara, Ireland.
Welch, Terry, "*A Technique for High-Performance Data Compression*", 1984, Sperry Research Center, IEEE.
Zeletin, R. et al., "*Applying Location Aware Computing for Electronic Commerce: Mobile Guide*", Oct. 20-22, 1998, 5th Conference on Computer Communications, AFRICOM-CCDC '98, Tunis.
Zell, Markus, "*Selection of Converter Chains by Means of Quality of Service Analysis*", Thesis, Feb. 12, 1998, Technische Universitat Berlin.
Implicit Networks, Inc. v. *Advanced Micro Devices, Inc. et al.*; C08-0184 JLR; USDC for the Western District of Washington, Seattle Division.
Feb. 4, 2008 Plaintiff's Original Complaint.
Aug. 26, 2008 Defendant NVIDIA Corporation's Answer to Complaint.
Aug. 26, 2008 Defendant Sun Microsystems, Inc.'s Answer to Complaint.
Aug. 27, 2008 Defendant Advanced Micro Devices, Inc.'s Answer to Complaint for Patent Infringement.
Aug. 27, 2008 RealNetworks, Inc.'s Answer to Implicit Networks, Inc.'s Original Complaint for Patent Infringement, Affirmative Defenses, and Counterclaims.
Aug. 27, 2008 Intel Corp.'s Answer, Defenses and Counterclaims.
Aug. 27, 2008 Defendant RMI Corporation's Answer to Plaintiff's Original Complaint.
Sep. 15, 2008 Plaintiff's Reply to NVIDIA Corporation's Counterclaims.
Sep. 15, 2008 Plaintiff's Reply to Sun Microsystems Inc.'s Counterclaims.
Sep. 16, 2008 Plaintiff's Reply to RealNetworks, Inc.'s Counterclaims.
Sep. 16, 2008 Plaintiff's Reply to Intel Corp.'s Counterclaims.
Dec. 10, 2008 Order granting Stipulated Motion for Dismissal with Prejudice re NVIDIA Corporation, Inc.
Dec. 16, 2008 Defendants AMD, RealNetworks, RMI, and Sun's Motion to Stay Pending the Patent and Trademark Office's Reexamination of the '163 Patent.
Dec. 29, 2008 Order granting Stipulated Motion for Dismissal without Prejudice of Claims re Sun Microsystems, Inc.
Jan. 5, 2009 Plaintiff's Opposition to Defendants AMD, RealNetworks, RMI, and Sun's Motion to Stay Pending Reexamination and Exhibit A.
Jan. 9, 2009 Reply of Defendants AMD, RealNetworks, RMI, and Sun's Motion to Stay Pending the Patent and Trademark Office's Reexamination of the '163 Patent.
Feb. 9, 2009 Order Granting Stay Pending the United States Patent and Trademark Office's Reexamination of U.S. Pat. No. 6,629,163.
Feb. 17, 2009 Order Granting Stipulated Motion for Dismissal of Advanced Micro Devices, Inc. with Prejudice.
May. 14, 2009 Order Granting Stipulated Motion for Dismissal of RMI Corporation with Prejudice.
Oct. 13, 2009 Order Granting Stipulated Motion for Dismissal of Claims Against and Counterclaims by Intel Corporation.
Oct. 30, 2009 Executed Order for Stipulated Motion for Dismissal of Claims Against and Counterclaims by RealNetworks, Inc.
Implicit Networks, Inc. v. *Microsoft Corp.*, C09-5628 HLR; USDC for the Northern District of California, San Francisco Division.
Nov. 30, 2009 Plaintiff's Original Complaint, *Implicit* v *Microsoft*, Case No. 09-5628.
Jan. 22, 2010 Order Dismissing Case, *Implicit* v *Microsoft*, Case No. 09-5628.
Implicit Networks, Inc. v. *Cisco Systems, Inc.*, C10-3606 HRL; USDC for the Northern District of California, San Francisco Division.
Aug. 16, 2010 Plaintiff's Original Complaint, *Implicit* v *Cisco*, Case No. 10-3606.
Nov. 22, 2010 Defendant Cisco Systems, Inc.'s Answer and Counterclaims, *Implicit* v *Cisco*, Case No. 10-3606.
Dec. 13, 2010 Plaintiff, Implicit Networks, Inc.'s, Answer to Counterclaims, *Implicit* v *Cisco*, Case No. 10-3606.
Oct. 4, 2011 Order of Dismissal with Prejudice, *Implicit* v *Cisco*, Case No. 10-3606.
Implicit Networks, Inc. v. *Citrix Systems, Inc.*, C10-3766 JL; USDC for the Northern District of California, San Francisco Division.
Oct. 24, 2010 Plaintiff's Original Complaint, *Implicit* v *Citrix*, Case No. 10-3766.
Dec. 1, 2010 Plaintiff's First Amended Complaint, *Implicit* v *Citrix*, Case No. 10-3766.
Jan. 14, 2011 Degendant Citrix Systems, Inc.'s Answer, Defenses and Counter-complaint for Declaratory Judgment, *Implicit* v *Citrix*, Case No. 10-3766.

(56) **References Cited**

OTHER PUBLICATIONS

Feb. 18, 2011 Plaintiff, Implicit Networks, Inc.'s, Answer to Defendants Counterclaims, *Implicit* v *Citrix*, Case No. 10-3766.
May 2, 2011 Order of Dismissal, *Implicit* v *Citrix*, Case No. 10-3766.
Implicit Networks, Inc. v.*F5 Networks, Inc.*, C10-3365 JCS; USDC for the Northern District of California, San Francisco Division.
Jul. 30, 2010 Plaintiff's Original Complaint, *Implicit* v *F5*, Case No. 10-3365.
Oct. 13, 2010 Defendants' Answer and Counter-Complaint, *Implicit* v *F5*, Case No. 10-3365.
Nov. 3, 2010 Plaintiff's Answer to Counter-Complaint, *Implicit* v *F5*, Case No. 10-3365.
Dec. 10, 2010 Plaintiff's First Amended Complaint, *Implicit* v *F5*, Case No. 10-3365.
Jan. 14, 2011 Defendants' Answer to 1st Amended Complaint and Counterclaim, *Implicit* v *F5*, Case No. 10-3365.
Feb. 18, 2011 Plaintiff's Answer to F5' s Amended Counter-Complaint, *Implicit* v *F5*, Case No. 10-3365.
Apr. 18, 2011 Defendants' Amended Answer to 1st Amended Complaint and Counter-Complaint, *Implicit* v *F5*, Case No. 10-3365.
May 5, 2011 Plaintiff's Answer to F5' s Amended Counter-Complaint, *Implicit* v *F5*, Case No. 10-3365.
Jul. 22, 2011 F5 Networks, Inc.'s Invalidity Contentions, *Implicit* v *F5*, Case No. 10-3365.
Jul. 22, 2011 F5 Networks, Inc.'s Invalidity Contentions, Exhibit A, *Implicit* v *F5*, Case No. 10-3365 (31 documents).
Jul. 22, 2011 F5 Networks, Inc.'s Invalidity Contentions, Exhibit B, *Implicit* v *F5*, Case No. 10-3365.
Oct. 18, 2011 Joint Claim Construction & Pre-Hearing Statement (PR 4-3), *Implicit* v *F5*, Case No. 10-3365.
Oct. 18, 2011 Joint Claim Construction & Pre-Hearing Statement (PR 4-3) Exhibit A, *Implicit* v *F5*, Case No. 10-3365 (2 documents).
Nov. 28, 2011 Plaintiff's Opening Claim Construction Brief, *Implicit* v *F5*, Case No. 10-3365.
Nov. 29, 2011 Amended Joint Claim Construction & Pre-Hearing Statement, *Implicit* v *F5*, Case No. 10-3365.
Nov. 29, 2011 Amended Joint Claim Construction & Pre-Hearing Statement, Exhibit A, *Implicit* v *F5*, Case No. 10-3365.
Dec. 12, 2011 Defendants' Claim Construction Brief, *Implicit* v *F5*, Case No. 10-3365.
Dec. 19, 2011 Plaintiff's Reply to Defendants' (F5, HP, Juniper) Responsive Claim Construction Brief (4-5), *Implicit* v *F5*, Case No. 10-3365.
Jan. 27, 2012 Transcript of Proceeding Held on Jan. 17, 2012; *Implicit* v *F5*, Case No. 10-3365.
Jan. 27, 2012 Transcript of Proceeding Held on Jan. 18, 2012; *Implicit* v *F5*, Case No. 10-3365.
Jan. 27, 2012 Transcript of Proceeding Held on Jan. 19, 2012; *Implicit* v *F5*, Case No. 10-3365.
Feb. 29, 2012 Claim Construction Order.
Aug. 15, 2012 Storer Invalidity Report.
Sep. 10, 2012 Implicit's Expert Report of Scott M. Nettles.
Mar. 13, 2013 Order Granting Defendants' Motion for Summary Judgment.
Apr. 9, 2013 Notice of Appeal to the Federal Circuit.
Implicit Networks, Inc. v. *Hewlett-Packard Company*, C10-3746 JCS: USDC for the Northern District of California, San Francisco Division.
Aug. 23, 2010 Plaintiff's Original Complaint, *Implicit* v *HP*, Case No. 10-3746.
Nov. 23, 2010 Plaintiff's First Amended Complaint, *Implicit* v *HP*, Case No. 10-3746.
Jan. 14, 2011 Defendant HP's Answer and Counterclaims, *Implicit* v *HP*, Case No. 10-3746.
Feb. 18, 2011 Implicit Networks, Inc.'s Answer to HP Counterclaims, *Implicit* v *HP*, Case No. 10-3746.
May 10, 2011 Plaintiff's Amended Disclosure of Asserted Claims and Infringement Contentions, Case No. 10-3746.
Jun. 30, 2011 Defendant HP Company's Invalidity Contentions, *Implicit* v *HP*, Case No. 10-3746.
Jun. 30, 2011 Defendant HP Company's Invalidity Contentions, A1-14, *Implicit* v *HP*, Case No. 10-3746.
Jun. 30, 2011 Defendant HP Company's Invalidity Contentions, B1-21, *Implicit* v *HP*, Case No. 10-3746.
Implicit Networks, Inc. v. Juniper Networks, C10-4234 EDL: USDC for the Northern District of California, San Francisco Division.
Sep. 20, 2010 Plaintiff's Original Complaint, *Implicit* v *Juniper* , Case No. 10-4234.
Nov. 12, 2012 Juniper Network's Motion to Dismiss for Failure to State a Claim Under Rule 12(B)(6): Memorandum of Points and Authorities; *Implicit* v *Juniper*, Case No. 10-4234.
Nov. 12, 2010 Juniper Network's Request for Judicial Notice in Support of its Motion to Dismiss for Failure to State a Claim Under Rule 12(B)(6): Memorandum of Points and Authorities; *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 1, 2010 First Amended Complaint; *Implicit* v *Juniper*, Case No. 10-4234.
Jan. 18, 2011 Juniper Networks, Inc.'s Answer and Affirmative Defenses to 1st Amended Complaint, *Implicit* v *Juniper*, Case No. 10-4234.
Feb. 18, 2011 Plaintiff's Answer to Defendant's Counterclaims, *Implicit* v *Juniper*, Case No. 10-4234.
May 23, 2011 Plaintiff's Disclosure of Asserted Claims and Infringement Contentions, *Implicit* v *Juniper*, Case No. 10-4234.
Nov. 15, 2011 Plaintiff's Amended Disclosure of Asserted Claim and Infringement Contentions, *Implicit* v *Juniper*, Case No. 10-4234.
Nov. 28, 2011 Spencer Hosie Declaration in Support of Plaintiffs Opening Claim Construction Brief), *Implicit* v *Juniper*, Case No. 10-4234.
Nov. 28, 2011 Spencer Hosie Declaration in Support of Plaintiffs Opening Claim Construction Brief Exhibit E, *Implicit* v *Juniper*, Case No. 10-4234.
Nov. 28, 2011 Spencer Hosie Declaration in Support of Plaintiffs Opening Claim Construction Brief Exhibit J, *Implicit* v *Juniper*, Case No. 10-4234.
Nov. 28, 2011 Spencer Hosie Declaration in Support of Plaintiffs Opening Claim Construction Brief Exhibit K, *Implicit* v *Juniper*, Case No. 10-4234.
Nov. 28, 2011 Spencer Hosie Declaration in Support of Plaintiff's Opening Claim Construction Brief Exhibits M-O, *Implicit* v. *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit B, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit F, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit N, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit P, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit Q, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit S., *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit T-1, *Implicit* v *Juniper*, Case. No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit T-2, *Implicit* v *Juniper*, Case. No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit T-3, *Implicit* v *Juniper*, Case. No. 10-4234.

(56) **References Cited**

OTHER PUBLICATIONS

Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit T-4, *Implicit* v *Juniper*, Case. No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit U, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit V, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit W, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants's Claim Construction Brief, Exhibit X, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit Y-1, *Implicit* v *Juniper*1, Case. No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit Y-2, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit Y-3, *Implicit* v *Juniper*, Case. No. 19-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit Y-4, *Implicit* v *Juniper*, Case. No. 10-4234.
Dec. 12, 2011 Holly Hogan Declaration in Support of Defendants' Claim Construction Brief, Exhibit Z, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 19, 2011 Spencer Hosie Declaration in Support of Plaintiff's Reply Claim Construction Brief, *Implicit* v *Juniper*, Case No. 10-4234.
Dec. 19, 2011 Spencer Hosie Declaration in Support of Plaintiff's Reply Claim Construction Brief, Exhibit P, *Implicit* v *Juniper*, Case No. 10-4234.
Jan. 10, 2012 Plaintiff's Jan. 10, 2012 Amended Disclosure of Asserted Claims and Infringement Contentions, *Implicit* v *Juniper*, Case No. 10-4234.
Feb. 10, 2012 Juniper Networks, Inc.'s Supplemental Invalidity Contentions, *Implicit* v *Juniper*, Case No. 10-4234.
Feb. 10, 2012 Juniper Networks, Inc.'s Supplemental Invalidity Contentions, Exhibit A1 *Implicit* v *Juniper*, Case No. 10-4234.
Feb. 10, 2012 Juniper Networks, Inc.'s Supplemental Invalidity Contentions, Exhibit A2, *Implicit* v *Juniper*, Case No. 10-4234.
Feb. 10, 2012 Juniper Networks, Inc.'s Supplemental Invalidity Contentions, Exhibit A3, *Implicit* v *Juniper*, Case No. 10-4234.
Feb. 10, 2012 Juniper Networks, Inc.'s Supplemental Invalidity Contentions, Exhibit A4, *Implicit* v *Juniper*, Case No. 10-4234.
Feb. 10, 2012 Juniper Networks, Inc.'s Supplemental Invalidity Contentions, Exhibit B1, *Implicit* v *Juniper*, Case No. 10-4234.
Feb. 29, 2012 Plaintiff's Feb. 29, 2012 Amended Disclosure of Asserted Claims and Infringement Contentions, *Implicit* v *Juniper*, Case No. 10-4234.
Apr. 6, 2012 Plaintiff's Apr. 6, 2012 Amended Disclosure of Asserted Claims and Infringement Contentions, *Implicit* v *Juniper*, Case No. 10-4234.
Apr. 9, 2012 Plaintiff's Apr. 9, 2012 Amended Disclosure of Asserted Claims and Infringement Contentions, *Implicit* v *Juniper*, Case No. 10-4234.
Sep. 11, 2012 Implicit's Expert Report of Scott Nettles.
Nov. 9, 2012 Juniper's Notice of Motion and Memorandum of Law ISO Motion for Summary Judgment or, in the alternative, for Partial Summary Judgment, on the Issue of Invalidity.
Nov. 9, 2012 Exhibit 2 to Declaration in support of Juniper's Motion for Summary Judgment—Calvert Expert Report.
Nov. 9, 2012 Exhibit 3 to Declaration in support of Juniper's Motion for Summary Judgment—Calvert Supplemental Expert Report.
Nov. 26, 2012 Implicit Opposition to Juniper's and F5 Motion on Invalidity.
Nov. 26, 2012 Exhibit A to Hosie Declaration—Aug. 27, 2012 Excerpts from David Blaine deposition.
Nov. 26, 2012 Exhibit B to Hosie Declaration—Oct. 25, 2012 Excerpts from Kenneth Calvert Deposition.
Nov. 26, 2012 Exhibit C to Hosie Declaration—Aug. 15, 2012 Excerpts from Kenneth Calvert Expert Report.
Nov. 26, 2012 Exhibit D to Hosie Declaration—U.S. Pat. No. 6,651,099 to Dietz et al.
Nov. 26, 2012 Exhibit E to Hosie Declaration—Understanding Packet-Based and Flow-Based Forwarding.
Nov. 26, 2012 Exhibit F to Hosie Declaration—Wikipedia on Soft State.
Nov. 26, 2012 Exhibit G to Hosie Declaration—Sprint Notes.
Nov. 26, 2012 Exhibit H to Hosie Declaration—Implicit's Supplemental Response to Juniper's 2nd Set of Interrogatories.
Nov. 26, 2012 Exhibit I to Hosie Declaration—U.S. Pat. No. 7,650,634 (Zuk).
May 13, 2013 Order Granting Defendants' Motion for Summary Judgment.
Other Implicit Networks, Inc. Prosecution Matters.
U.S. Appl. No. 11/933,022 Utility Application filed Oct. 31, 2007.
U.S. Appl. No. 11/933,022 Preliminary Amendment filed Feb. 19, 2008.
U.S. Appl. No. 11/933,022 Office Action dated Jun. 24, 2009.
U.S. Appl. No. 11/933,022 Amendment filed Sep. 24, 2009.
U.S. Appl. No. 11/933,022 Office Action dated Dec. 11, 2009.
U.S. Appl. No. 11/933,022 Amendment and Response dated Jan. 29, 2010.
U.S. Appl. No. 11/933,022 Notice of Allowance dated Mar. 2, 2010.
U.S. Appl. No. 11/933,022 Issue Notification dated May 4, 2010.
U.S. Appl. No. 10/636,314 Utility Application filed Aug. 6, 2003.
U.S. Appl. No. 10/636,314 Office Action dated Apr. 7, 2008.
U.S. Appl. No. 10/636,314 Response to Restriction Requirement dated Aug. 5, 2008.
U.S. Appl. No. 10/636,314 Office Action dated Oct. 3, 2008.
U.S. Appl. No. 10/636,314 Response to Office Action dated Apr. 3, 2009.
U.S. Appl. No. 10/636,314 Notice of Non-Compliant Amendment dated May 4, 2009.
U.S. Appl. No. 10/636,314 Amendment to Office Action Response dated Jun. 4, 2009.
U.S. Appl. No. 10/636,314 Notice of Non-Compliant Amendment dated Jun. 12, 2009.
U.S. Appl. No. 10/636,314 Amendment to Office Action dated Jul. 10, 2009.
U.S. Appl. No. 10/636,314 Final Rejection Office Action dated Oct. 21, 2009.
U.S. Appl. No. 10/636,314 Amendment after Final Office Action dated Dec. 14, 2009.
U.S. Appl. No. 10/636,314 Advisory Action dated Jan. 11, 2010.
U.S. Appl. No. 10/636,314 Notice of Non-Compliant Amendment dated Jan. 11, 2010.
U.S. Appl. No. 10/636,314 Supplemental Amendment and Response dated Mar. 13, 2010.
U.S. Appl. No. 10/636,314 Office Action dated May 11, 2010.
U.S. Appl. No. 10/636,314 Amendment and Response dated Sep. 13, 2010.
U.S. Appl. No. 10/636,314 Final Rejection dated Nov. 24, 2010.
U.S. Appl. No. 10/636,314 Notice of Appeal dated May 19, 2011.
U.S. Appl. No. 10/636,314 Amendment and Request for Continued Examination dated Jul. 19, 2011.
U.S. Appl. No. 10/636,314 Notice of Allowance dated Sep. 13, 2011.
U.S. Appl. No. 10/636,314 Notice of Allowance dated Sep. 19, 2011.
U.S. Appl. No. 10/636,314 Issue Notification dated Oct. 19, 2011.
U.S. Appl. No. 09/474,664 Utility Application filed Dec. 29, 1999.
U.S. Appl. No. 09/474,664 Office Action dated Sep. 23, 2002.
U.S. Appl. No. 09/474,664 Amendment and Response dated Feb. 24, 2003.

(56) **References Cited**

OTHER PUBLICATIONS

U.S. Appl. No. 09/474,664 Notice of Allowance dated May 20, 2003.
U.S. Appl. No. 90/010,356 Request for Ex Parte Reexamination dated Dec. 15, 2008.
U.S. Appl. No. 90/010,356 Office Action Granting Reexamination dated Jan. 17, 2009.
U.S. Appl. No. 90/010,356 First Office Action dated Jul. 7, 2009.
U.S. Appl. No. 90/010,356 First Office Action Response dated Sep. 1, 2009.
U.S. Appl. No. 90/010,356 Patent Owner Interview Summary dated Oct. 23, 2009.
U.S. Appl. No. 90/010,356 Office Action Final dated Dec. 4, 2009.
U.S. Appl. No. 90/010,356 Amendment and Response to Office Action dated Dec. 18, 2009.
U.S. Appl. No. 90/010,356 Amendment and Response to Office Action dated Jan. 4, 2010.
U.S. Appl. No. 90/010,356 Advisory Action dated Jan. 21, 2010.
U.S. Appl. No. 90/010,356 Amendment and Response to Advisory Action dated Feb. 8, 2010.
U.S. Appl. No. 90/010,356 Notice of Intent to Issue a Reexam Certificate dated Mar. 2, 2010.
U.S. Appl. No. 90/010,356 Reexamination Certificate Issued dated Jun. 22, 2010.
U.S. Appl. No. 95/000,659 Inter Partes Reexam Request dated Feb. 13, 2012.
U.S. Appl. No. 95/000,659 Order Granting Reexamination dated Apr. 3, 2012.
U.S. Appl. No. 95/000,659 Office Action dated Apr. 3, 2012.
U.S. Appl. No. 95/000,659 Office Action Response dated Jun. 4, 2012 (including Exhibits 1 & 2) (4 documents).
U.S. Appl. No. 95/000,659 Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012.
U.S. Appl. No. 95/000,659 Appendix R-1 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Declaration of Prof. Dr. Bernhard Plattner).
U.S. Appl. No. 95/000,659 Appendix R-2 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Prof. Dr. Bernhard Plattner CV).
U.S. Appl. No. 95/000,659 Appendix R-3 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Listing of Publications to Prof. Dr. Bernhard Plattner updated Feb. 2012).
U.S. Appl. No. 95/000,659 Appendix R-4 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012(Office Action Granting Reexamination in 95/000,660 dated May 10, 2012).
U.S. Appl. No. 95/000,659 Appendix R-5 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Office Action in 95/000,660 dated May 10, 2012).
U.S. Appl. No. 95/000,659 Appendix R-6 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Implicit Networks, Inc. U.S. Pat. No. 6,629,163 Claims Chart).
U.S. Appl. No. 95/000,659 Appendix R-7 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Internet Protocol DARPA Internet Program Protocol Specification dated Sep. 1991).
U.S. Appl. No. 95/000,659 Appendix R-8 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Atkinson, IP Encapsulating Security Payload (ESP) dated Aug. 1995).
U.S. Appl. No. 95/000,659 Appendix R-9 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Claim Construction Order dated Feb. 29, 2012).
U.S. Appl. No. 95/000,659 Appendix R-10-1 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (vol. I of Edward Balassanian Deposition Transcript dated May 30, 2012).
U.S. Appl. No. 95/000,659 Appendix R-10-2 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (vol. II of Edward Balassanian Deposition Transcript dated May 31, 2012).
U.S. Appl. No. 95/000,659 Appendix R-10-3 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (vol. III of Edward Balassanian Deposition Transcript dated Jun. 7, 2012).
U.S. Appl. No. 95/000,659 Appendix R-10-4 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (vol. IV of Edward Balassanian Deposition Transcript dated Jun. 8, 2012).
U.S. Appl. No. 95/000,659 Appendix R-11 to Third Party Comments to Patent Owner's Response to Office Action dated Jul. 5, 2012 (Implicit Networks, Inc.'s Response to Juniper Networks, Inc.'s First Set of Requests for Admission 1-32).
U.S. Appl. No. 95/000,659 Action Closing Prosecution dated Oct. 1, 2012.
U.S. Appl. No. 95/000,659 Petition to Withdraw and Reissue Action Closing Prosecution dated Nov. 20, 2012.
U.S. Appl. No. 95/000,659 Patent Owner Comments to Action Closing Prosecution dated Dec. 3, 2012.
U.S. Appl. No. 95/000,659 Opposition to Petition dated Dec. 17, 2012.
U.S. Appl. No. 95/000,659 Third Party Comments to Action Closing Prosecution dated Jan. 2, 2013.
U.S. Appl. No. 95/000,660 Inter Partes Reexam Request dated Mar. 2, 2012.
U.S. Appl. No. 95/000,660 Order Granting Reexamination dated May 10, 2012.
U.S. Appl. No. 95/000,660 Office Action dated May 10, 2012.
U.S. Appl. No. 95/000,660 Response to Office Action dated Jul. 10, 2012 (including Exhibits 1 and 2).
U.S. Appl. No. 95/000,660 Third Party Comments to Office After Patent Owner's Response dated Aug. 8, 2012 (including Revised Comments).
U.S. Appl. No. 95/000,660 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Declaration of Prof. Dr. Bernhard Plattner).
U.S. Appl. No. 95/000,660 Appendix R-1 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Prof. Dr. Bernhard Plattner CV).
U.S. Appl. No. 95/000,660 Appendix R-3 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Listing of Publications to Prof. Dr. Bernhard Plattner updated Feb. 2012).
U.S. Appl. No. 95/000,660 Appendix R-4 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012(Office Action Granting Reexamination in 95/000,660 dated May 10, 2012).
U.S. Appl. No. 95/000,660 Appendix R-5 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Office Action in 95/000,660 dated May 10, 2012).
U.S. Appl. No. 95/000,660 Appendix R-6 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Implicit Networks, Inc. U.S. Pat. No. 6,629,163 Claims Chart).
U.S. Appl. No. 95/000,660 Appendix R-7 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Internet Protocol DARPA Internet Program Protocol Specification dated Sep. 1991).
U.S. Appl. No. 95/000,660 Appendix R-8 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Atkinson, IP Encapsulating Security Payload (ESP) dated Aug. 1995).
U.S. Appl. No. 95/000,660 Appendix R-9 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Claim Construction Order dated Feb. 29, 2012).
U.S. Appl. No. 95/000,660 Appendix R-10 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (vol. I-IV of Edward Balassanian Deposition Transcript dated May 30, 2012).
U.S. Appl. No. 95/000,660 Appendix R-11 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8,

(56) **References Cited**

OTHER PUBLICATIONS

2012 (Shacham, A., et al, "*IP Payload Compression Protocol*", Network Working Group, RFC 3173 Sep. 2001).
U.S. Appl. No. 95/000,660 Appendix R-12 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Shacham, A., et al, "*IP Payload Compression Protocol*", Network Working Group, RFC 2393 Dec. 1998).
U.S. Appl. No. 95/000,660 Appendix R-13 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 ('163 Pfeiffer Claim Chart).
U.S. Appl. No. 95/000,660 Appendix R-14 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Ylonen, T., "*SSH Transport Layer Protocol*", Network Working Group—Draft Feb. 22, 1999).
U.S. Appl. No. 95/000,660 Appendix R-15 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Dommety, G., "*Key and Sequence Number Extensions to GRE*", Network Working Group, RFC 2890 Sep. 2000).
U.S. Appl. No. 95/000,660 Appendix R-16 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Monsour, R., et al, "*Compression in IP Security*" Mar. 1997).
U.S. Appl. No. 95/000,660 Appendix R-17 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012 (Friend, R., Internet Working Group RFC 3943 dated Nov. 2004 *Transport Layer Security Protocol Compression Using Lempel-Ziv-Stac*).
U.S. Appl. No. 95/000,660 Appendix R-18 to Third Party Comments to Patent Owner's Response to Office Action dated Aug. 8, 2012(Implicit Networks, Inc.'s Response to Juniper Networks, Inc.'s First Set of Requests for Admission 1-32).
U.S. Appl. No. 95/000,660 Revised—Third Party Comments to Office After Patent Owner's Response dated Nov. 2, 2012.
U.S. Appl. No. 95/000,660 Action Closing Prosecution dated Dec. 21, 2012.
U.S. Appl. No. 95/000,660 Comments to Action Closing Prosecution dated Feb. 21, 2013 (including Dec of Dr. Ng).
U.S. Appl. No. 95/000,660 Third Party Comments to Action Closing Prosecution dated Mar. 25, 2013.
PCT/US00/33634—PCT application (WO 01/2077 A2—dated Jul. 12, 2001).
PCT/US00/33634—Written Opinion (WO 01/50277 A3—dated Feb. 14, 2002).
PCT/US00/33634—International Search Report (dated Oct. 9, 2001).
PCT/US00/33634—Response to Official Communication dated Dec. 7, 2001 (dated Mar. 21, 2002).
PCT/US00/33634—International Preliminary Examination Report (dated Apr. 8, 2002).
PCT/US00/33634—Official Communication (dated Jan. 24, 2003).
PCT/US00/33634—Response to Official Communication dated Jan. 24, 2003 (dated Mar. 12, 2003).
PCT/US00/33634—Official Communication (dated May 13, 2004).
PCT/US00/33634—Response to Summons to Attend Oral Proceeding dated May 13, 2004 (dated Oct. 9, 2004).
PCT/US00/33634—Decision to Refuse a European Patent application (dated Nov. 12, 2004).
PCT/US00/33634—Minutes of the oral proceedings before the Examining Division (dated Oct. 12, 2004).
PCT/US00/33634—Closure of the procedure in respect to Application No. 00984234.5—2212 (dated Feb. 22, 2005).
May 3, 2013 Expert Report of Dr. Alfonso Cardenas Regarding Validity of U.S. Pat. Nos. 6,877,006; 7,167,864; 7,720,861; and 8,082,268 (6 documents).
Expert Report of Dr. Alfonso Cardenas Regarding Validity of U.S. Pat. No. 7,167,864 (3 documents).
"InfoReports User Guide: Version 3.3.1;" Platinum Technology, Publication No. PRO-X-331-UG00-00, printed Apr. 1998; pp. 1-430.

* cited by examiner

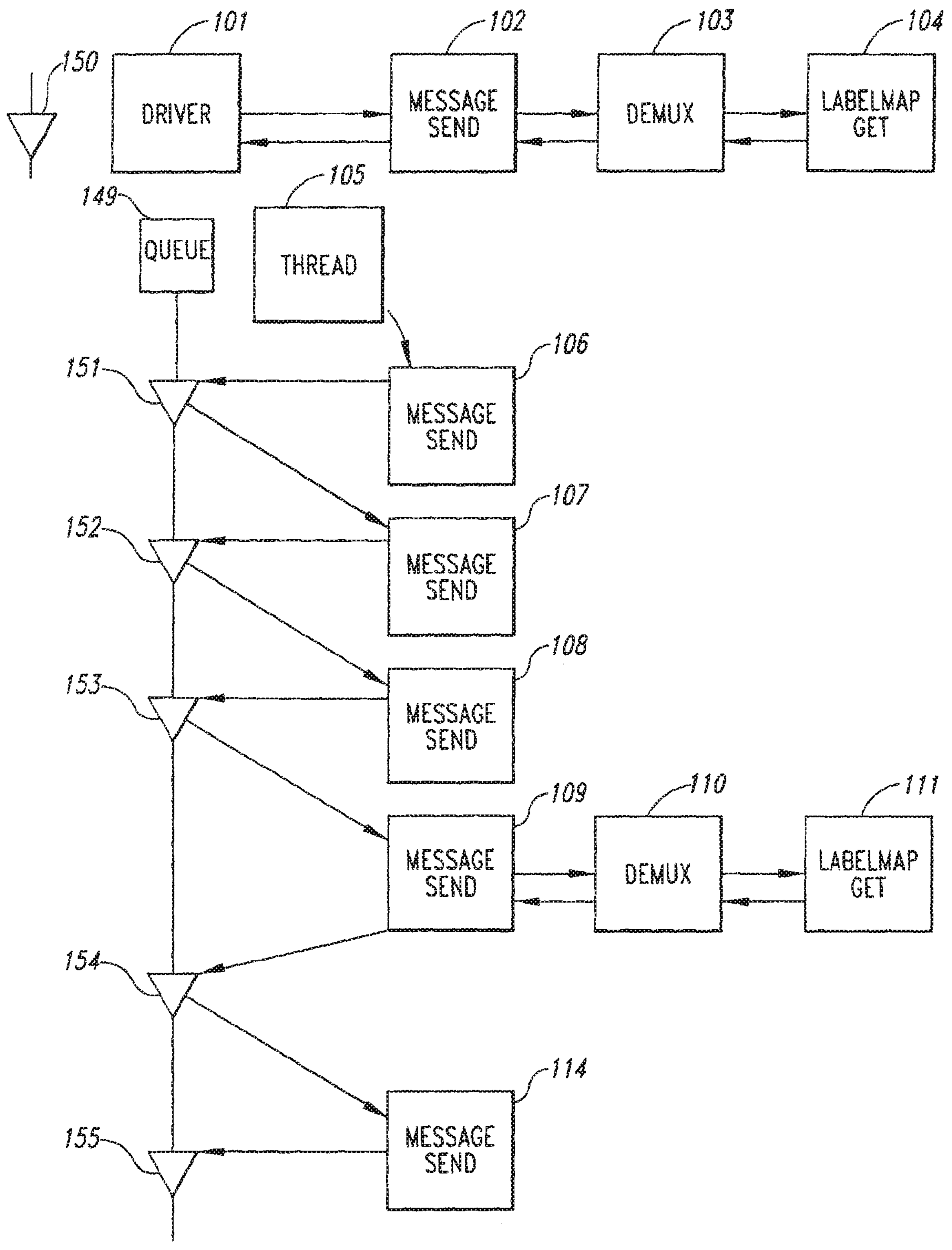

101
DRIVER
102
MESSAGE SEND
103
DEMUX
104
LABELMAP GET
150
149
QUEUE
105
THREAD
106
MESSAGE SEND
151
107
MESSAGE SEND
152
108
MESSAGE SEND
153
109
MESSAGE SEND
110
DEMUX
111
LABELMAP GET
154
114
MESSAGE SEND
155

Fig. 1

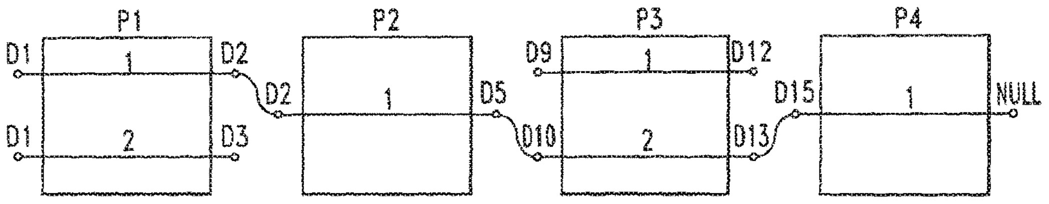

P1
P2
P3
P4
D1
1
D2
D1
2
D3
D2
1
D5
D9
1
D12
D10
2
D13
D15
1
NULL

Fig. 2

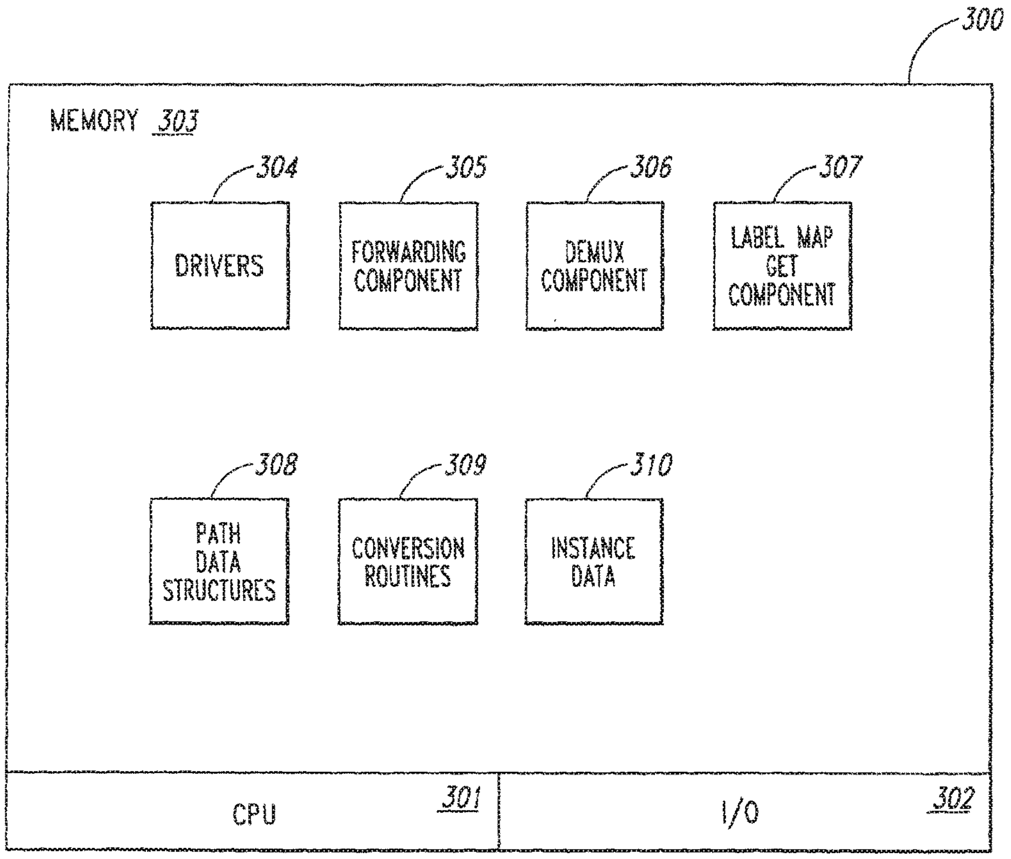

300
MEMORY 303
304
DRIVERS
305
FORWARDING COMPONENT
306
DEMUX COMPONENT
307
LABEL MAP GET COMPONENT
308
PATH DATA STRUCTURES
309
CONVERSION ROUTINES
310
INSTANCE DATA
CPU 301
I/O 302

Fig. 3

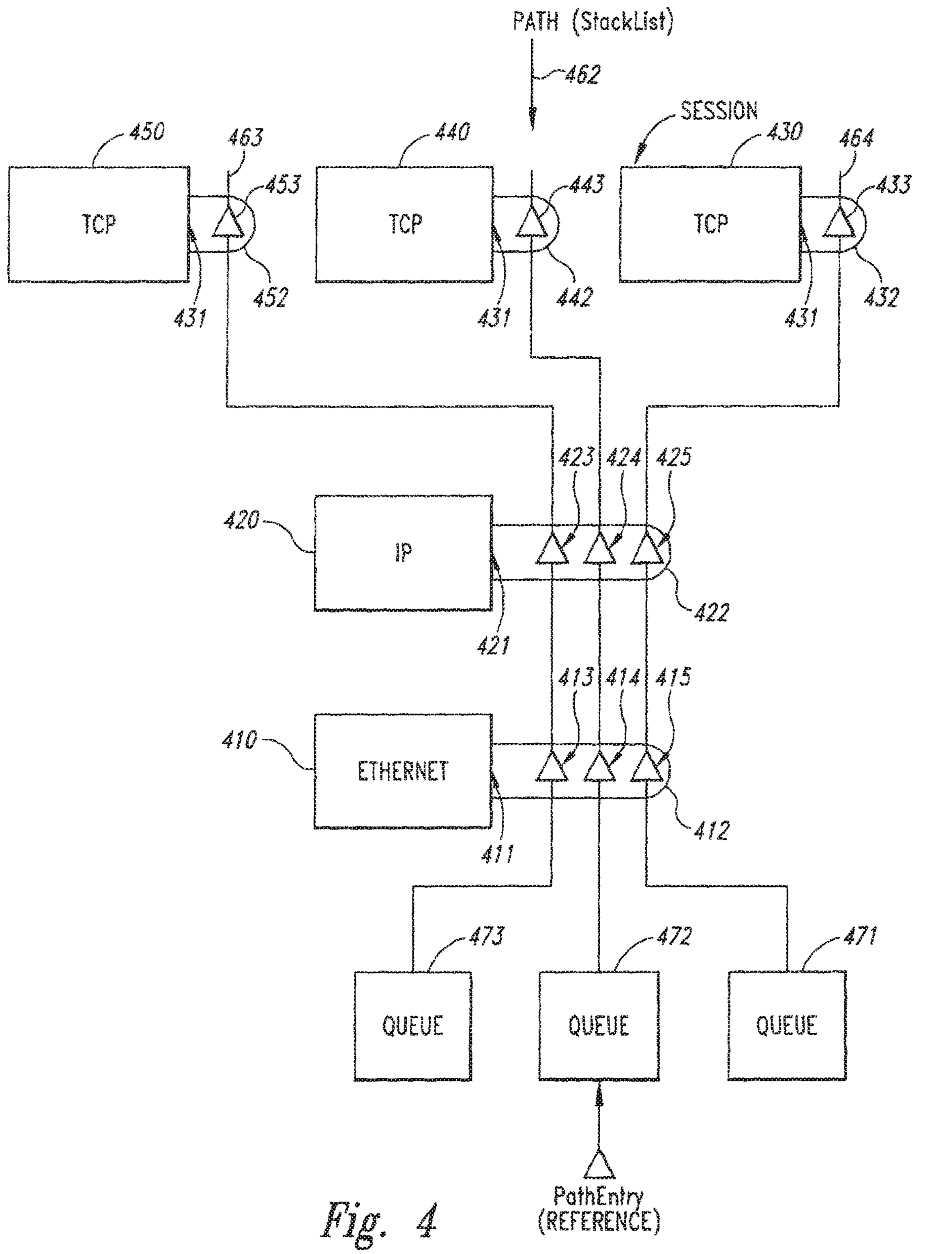
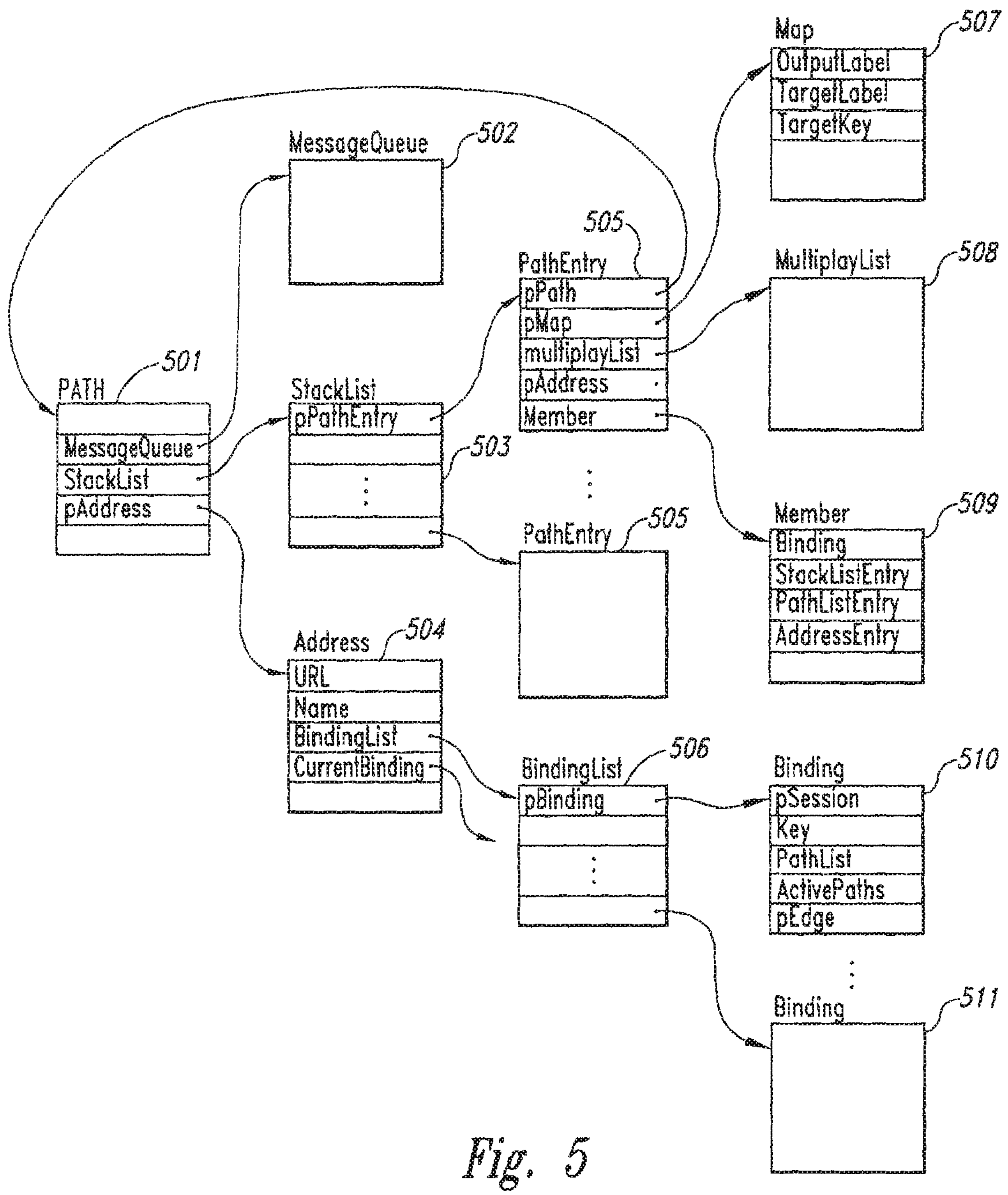

PATH (StackList)
462
SESSION
450
463
440
430
464
TCP
453
TCP
443
TCP
433
431
452
431
442
431
432
423
424
425
420
IP
421
422
413
414
415
410
ETHERNET
411
412
473
472
471
QUEUE
QUEUE
QUEUE
PathEntry
(REFERENCE)

Fig. 4



Map
507
OutputLabel
TargetLabel
TargetKey
MessageQueue
502
505
PathEntry
pPath
pMap
multiplayList
pAddress
Member
MultiplayList
508
PATH
501
MessageQueue
StackList
pAddress
StackList
pPathEntry
503
PathEntry
505
Member
509
Binding
StackListEntry
PathListEntry
AddressEntry
Address
504
URL
Name
BindingList
CurrentBinding
BindingList
506
pBinding
Binding
510
pSession
Key
PathList
ActivePaths
pEdge
Binding
511

Fig. 5

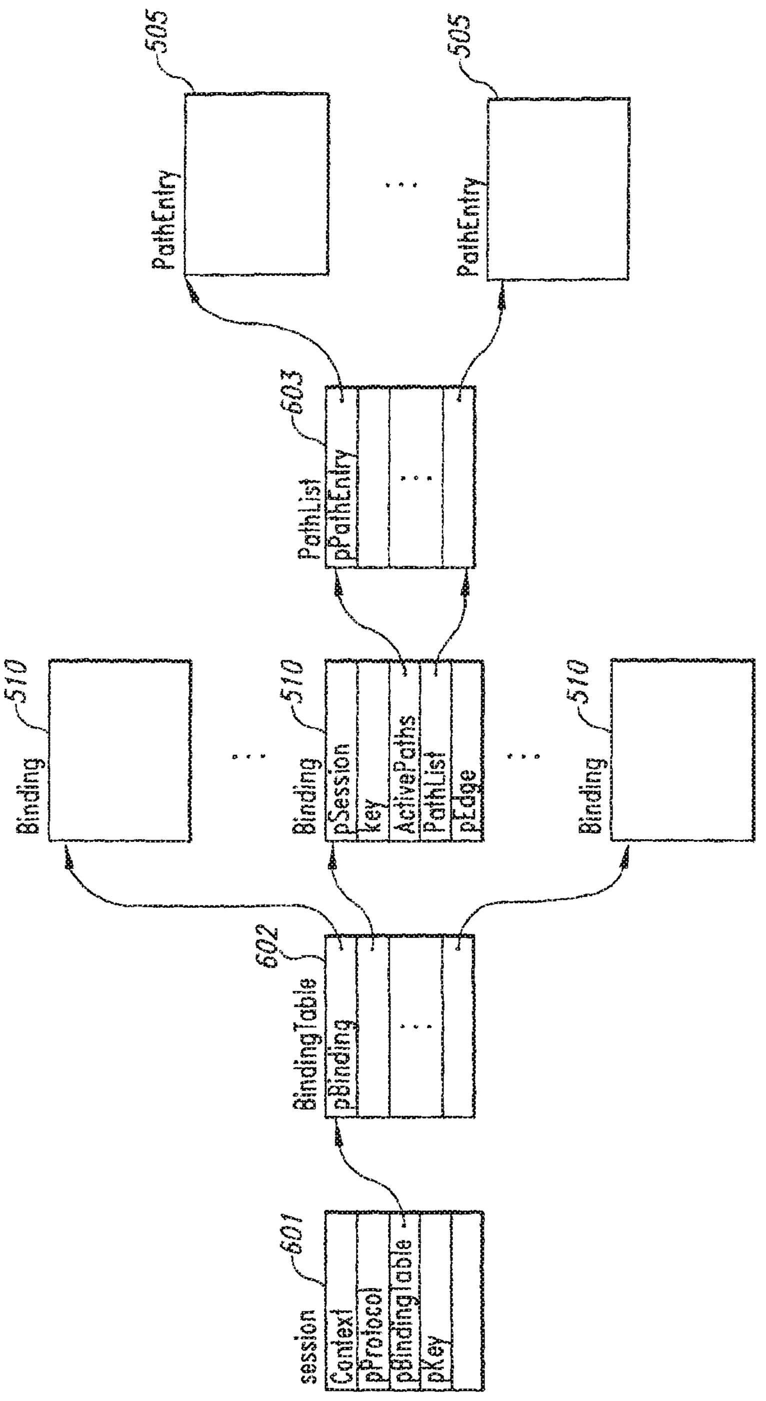

505
PathEntry
505
PathEntry
603
PathList
pPathEntry
510
Binding
510
Binding
pSession
key
ActivePaths
PathList
pEdge
510
Binding
602
BindingTable
pBinding
601
session
Context
pProtocol
pBindingTable
pKey

Fig. 6

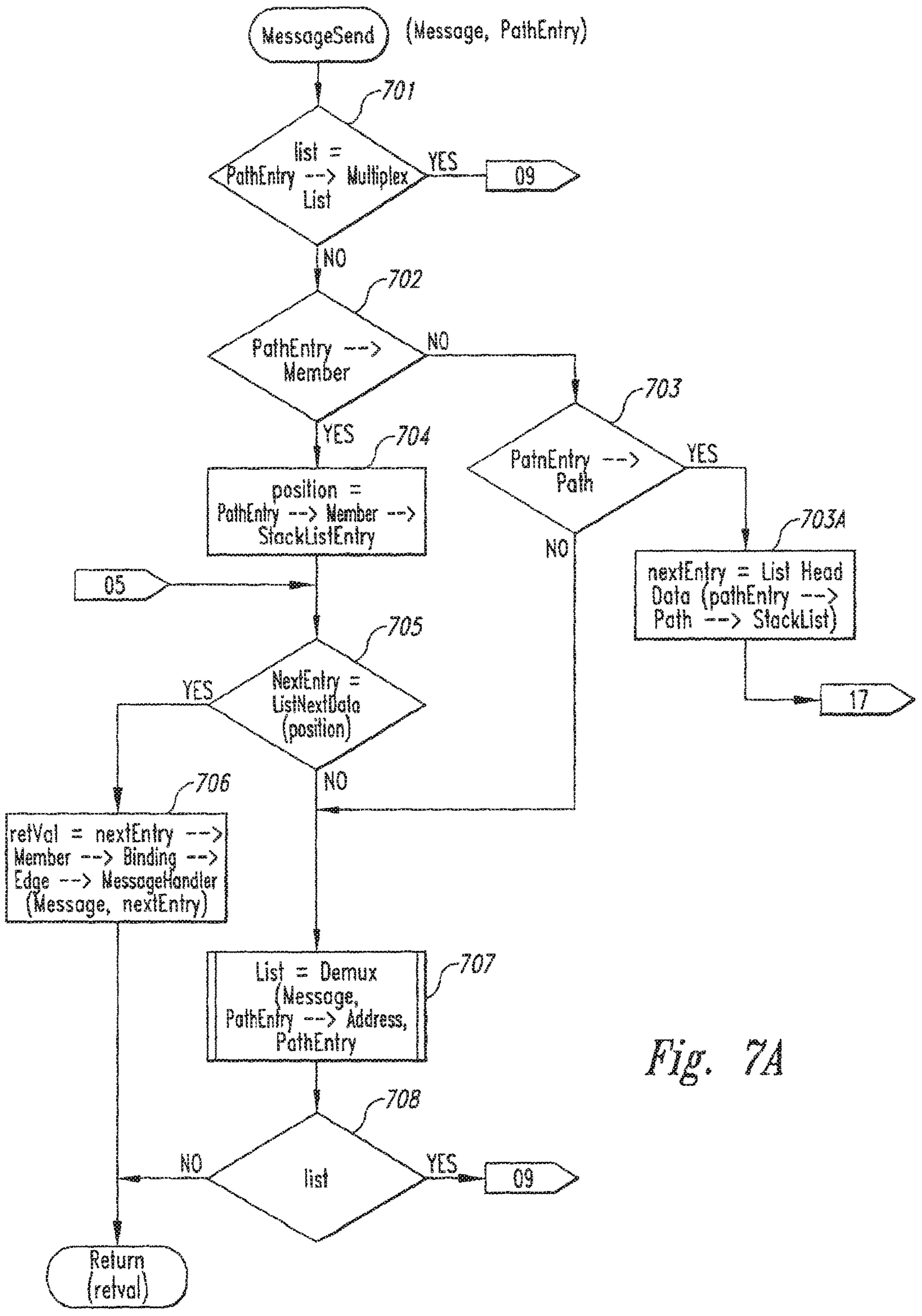

MessageSend (Message, PathEntry)
701
list = PathEntry --> Multiplex List
YES
09
NO
702
PathEntry --> Member
NO
YES
703
PatnEntry --> Path
YES
704
position = PathEntry --> Member --> StackListEntry
703A
nextEntry = List Head Data (pathEntry --> Path --> StackList)
NO
05
17
705
NextEntry = ListNextData (position)
YES
NO
706
retVal = nextEntry --> Member --> Binding --> Edge --> MessageHandler (Message, nextEntry)
707
List = Demux (Message, PathEntry --> Address, PathEntry
708
list
NO
YES
09
Return (retval)

Fig. 7A

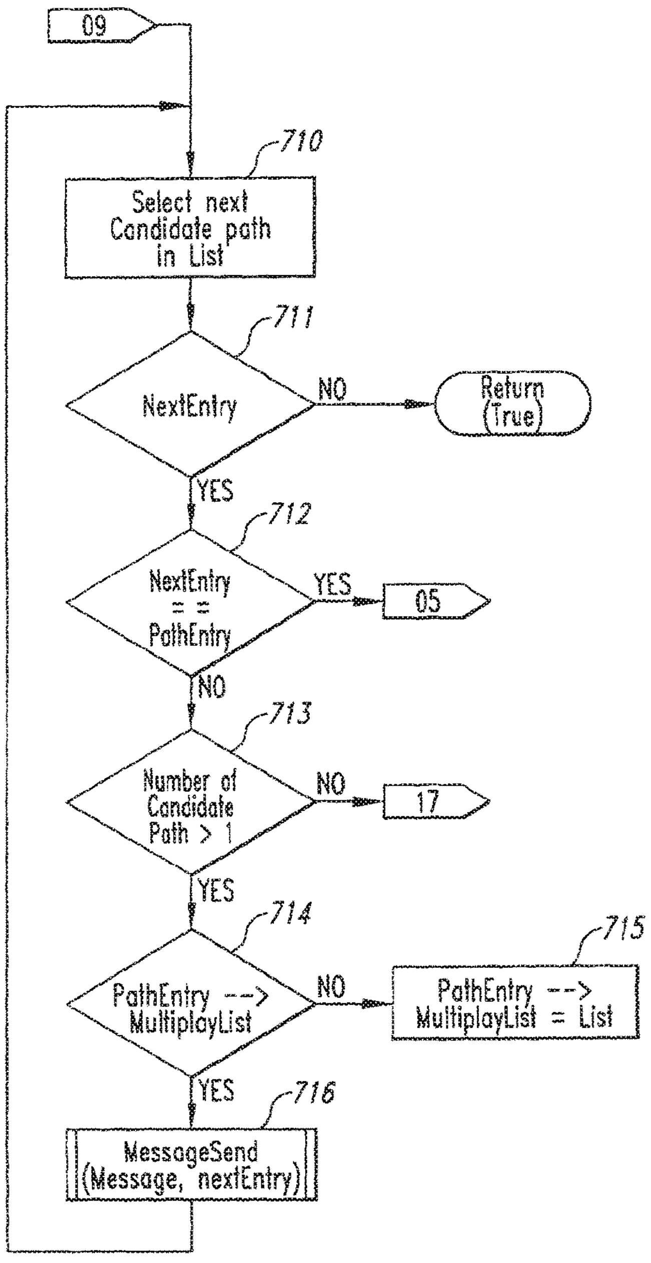

09
710
Select next Candidate path in List
711
NextEntry
NO
Return (True)
YES
712
NextEntry = = PathEntry
YES
05
NO
713
Number of Candidate Path > 1
NO
17
YES
714
PathEntry --> MultiplayList
NO
715
PathEntry --> MultiplayList = List
YES
716
MessageSend (Message, nextEntry)

Fig. 7B

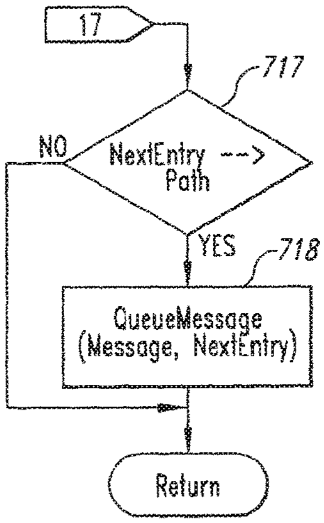

17
717
NextEntry --> Path
NO
YES
718
QueueMessage (Message, NextEntry)
Return

Fig. 7C

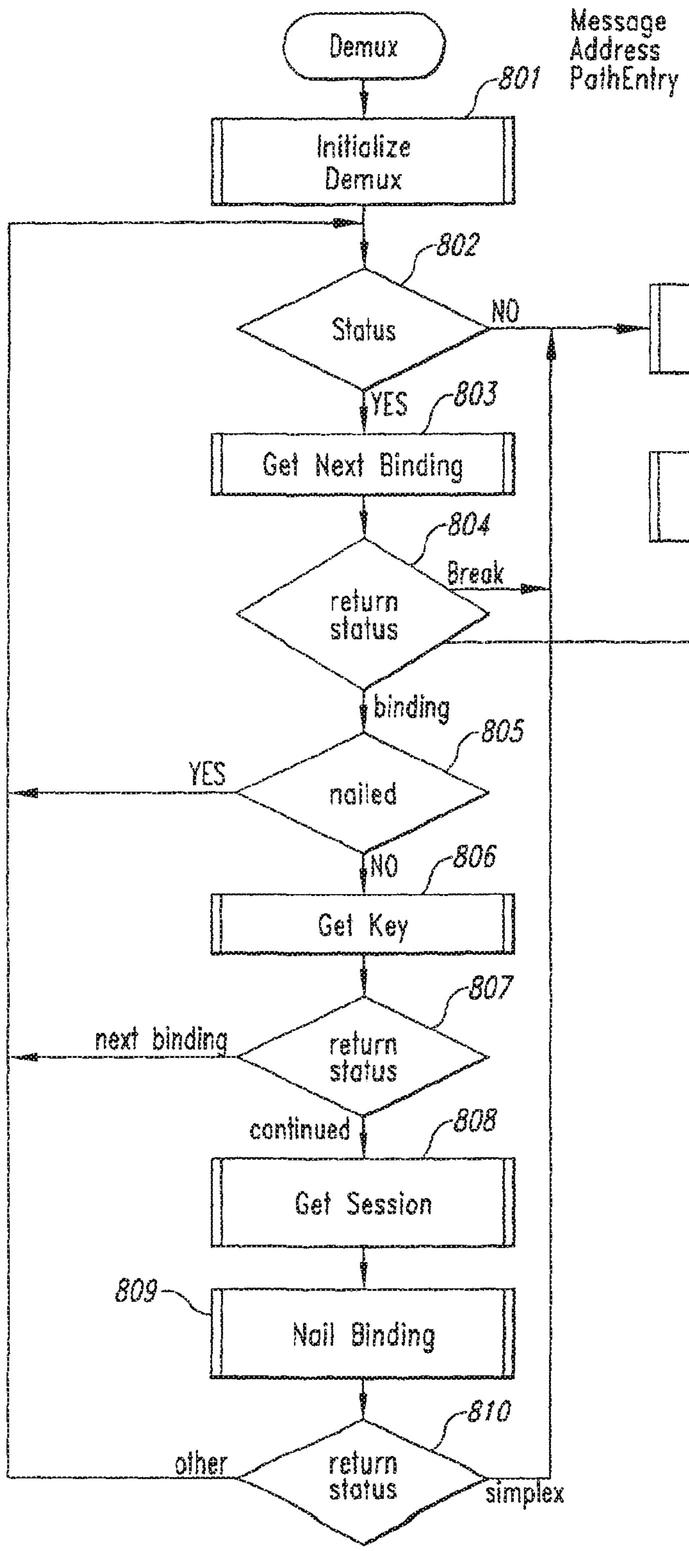

Demux
Message
Address
PathEntry
801
Initialize Demux
802
Status
NO
YES
811
Find Path
812
Process Path Hopping
803
Get Next Binding
804
return status
Break
multiple
binding
805
nailed
YES
NO
return
806
Get Key
807
return status
next binding
continued
808
Get Session
809
Nail Binding
810
return status
other
simplex

Fig. 8

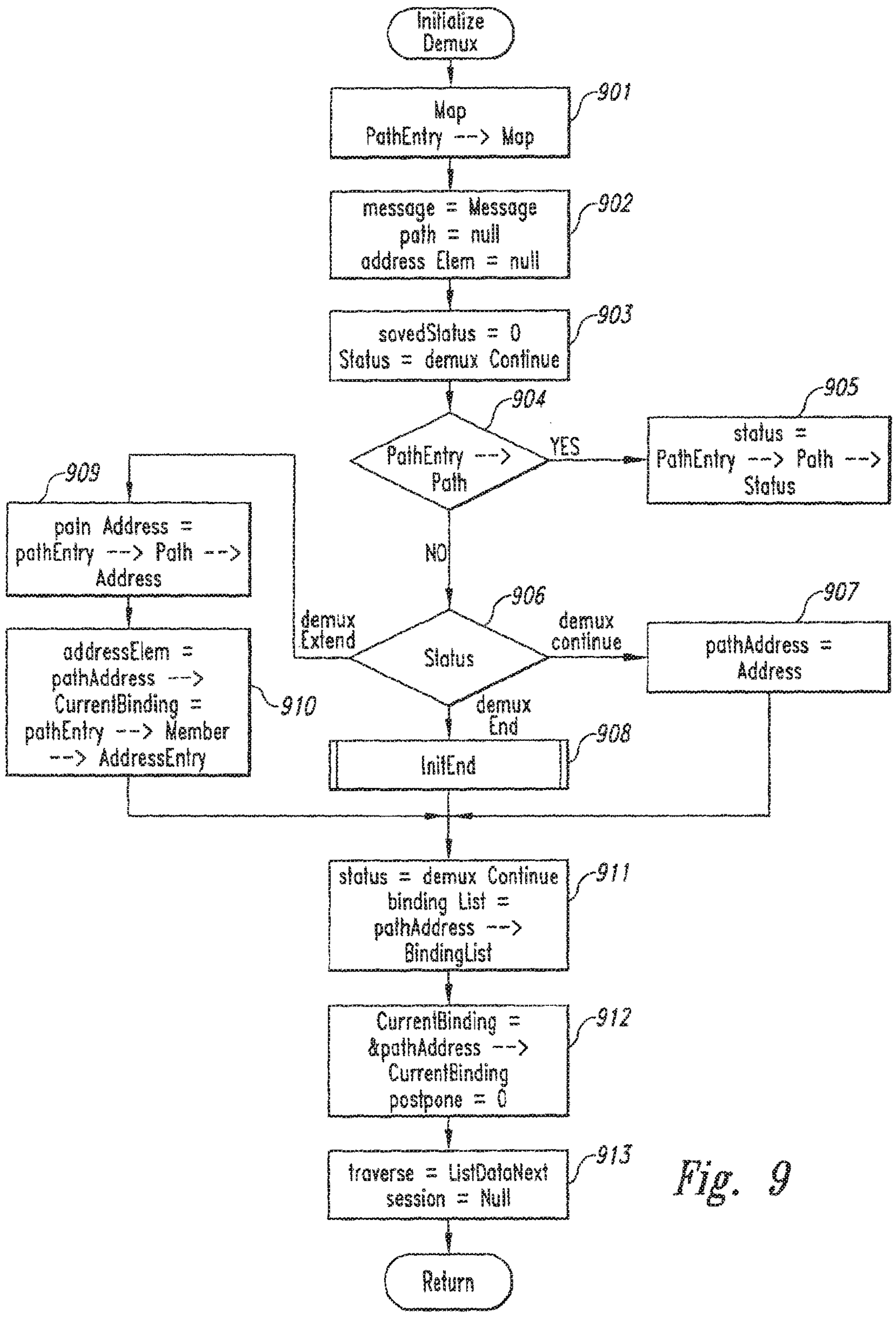

Initialize Demux
Map PathEntry --> Map
901
message = Message path = null address Elem = null
902
savedStatus = 0 Status = demux Continue
903
PathEntry --> Path
904
YES
status = PathEntry --> Path --> Status
905
NO
Status
906
demux Extend
demux continue
demux End
pathAddress = Address
907
InitEnd
908
path Address = pathEntry --> Path --> Address
909
addressElem = pathAddress --> CurrentBinding = pathEntry --> Member --> AddressEntry
910
status = demux Continue binding List = pathAddress --> BindingList
911
CurrentBinding = &pathAddress --> CurrentBinding postpone = 0
912
traverse = ListDataNext session = Null
913
Return

Fig. 9

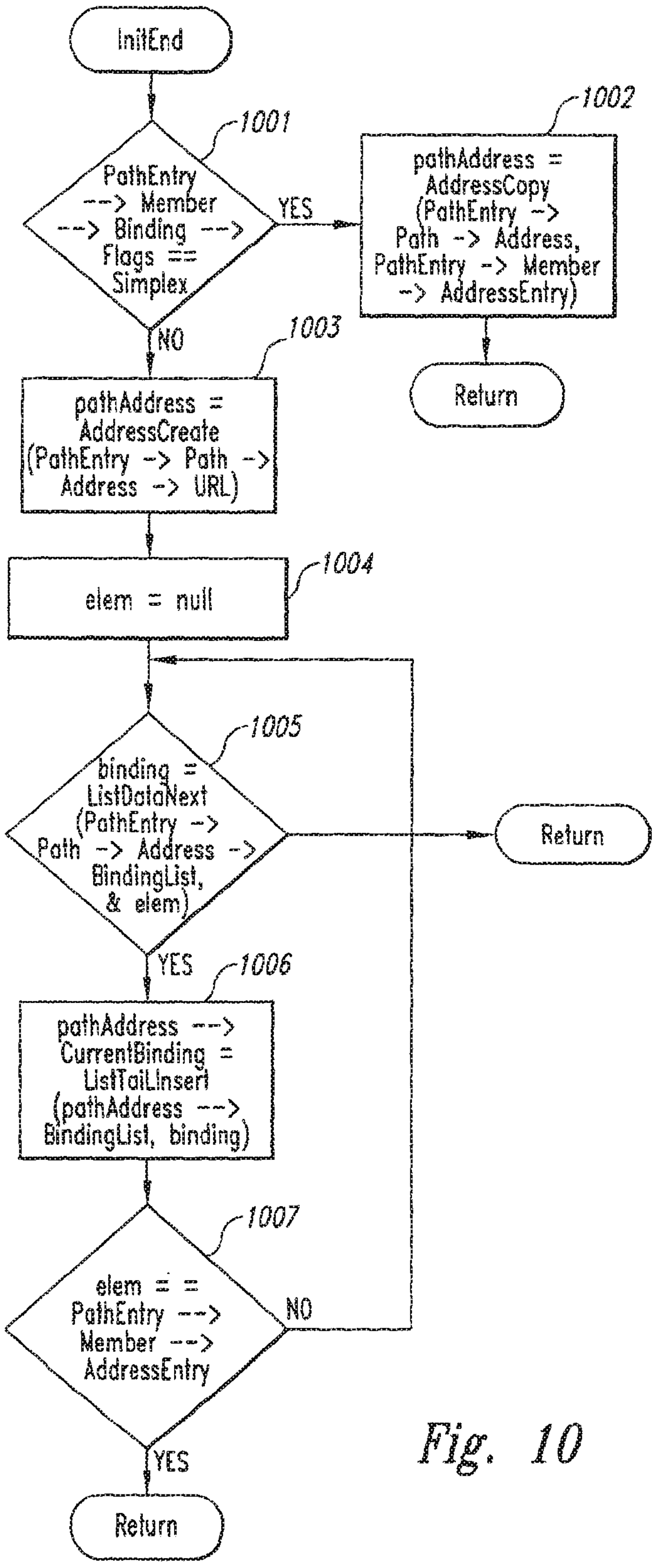

InitEnd
1001
PathEntry --> Member --> Binding --> Flags == Simplex
YES
1002
pathAddress = AddressCopy (PathEntry -> Path -> Address, PathEntry -> Member -> AddressEntry)
Return
NO
1003
pathAddress = AddressCreate (PathEntry -> Path -> Address -> URL)
1004
elem = null
1005
binding = ListDataNext (PathEntry -> Path -> Address -> BindingList, & elem)
Return
YES
1006
pathAddress --> CurrentBinding = ListTailInsert (pathAddress --> BindingList, binding)
1007
elem = = PathEntry --> Member --> AddressEntry
NO
YES
Return

Fig. 10

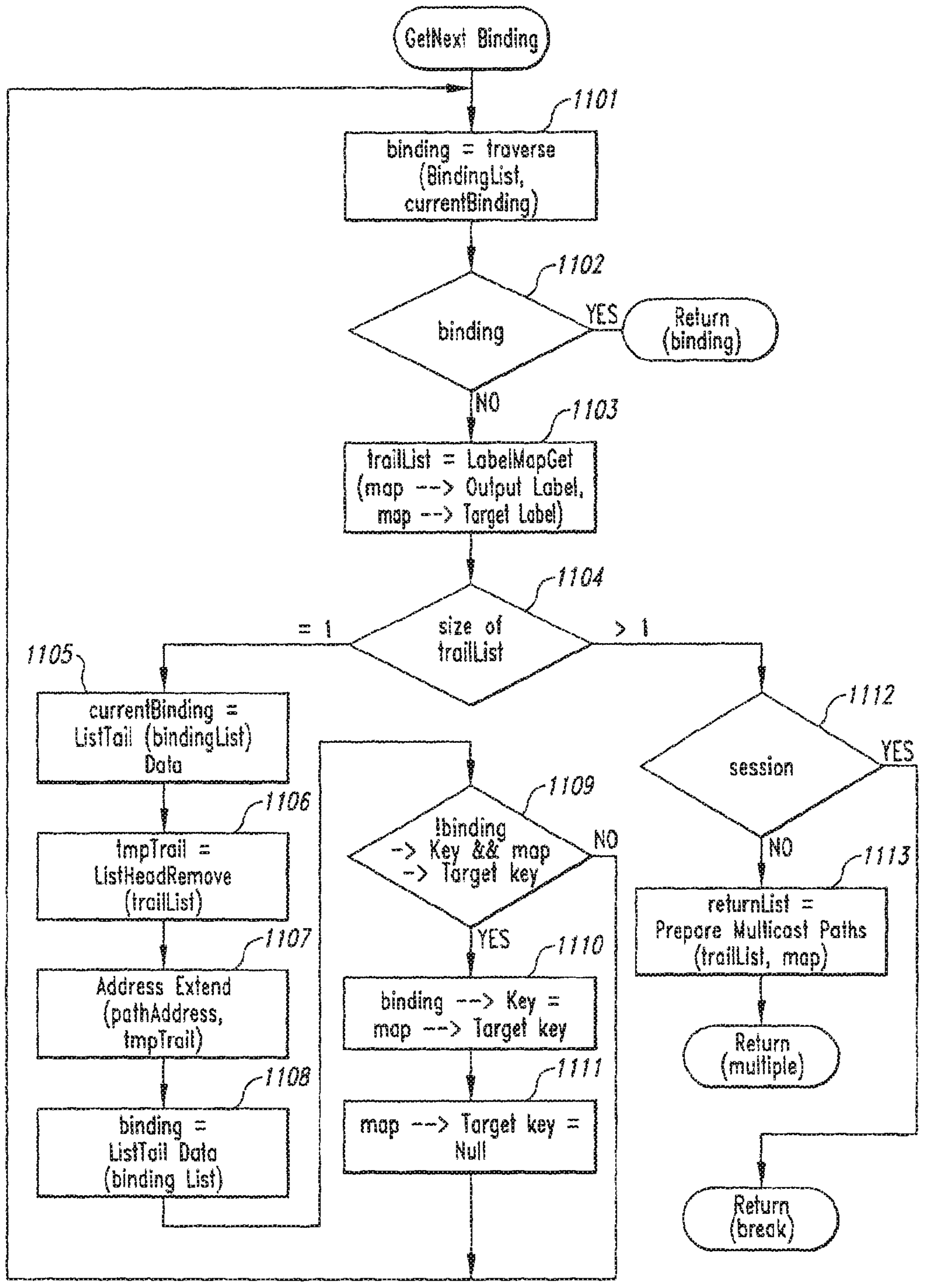
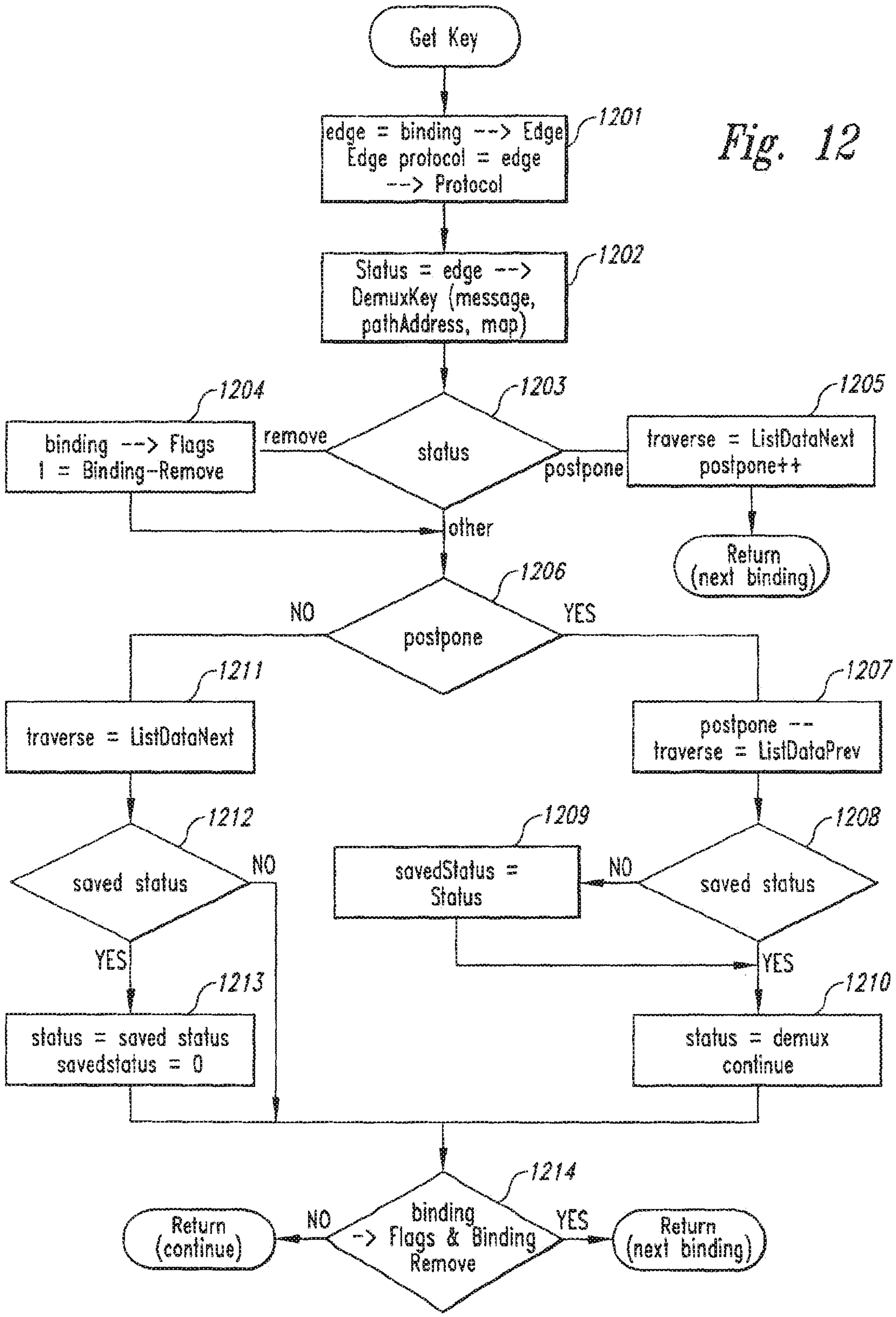

GetNext Binding
1101
binding = traverse (BindingList, currentBinding)
1102
binding
YES
Return (binding)
NO
1103
trailList = LabelMapGet (map --> Output Label, map --> Target Label)
1104
size of trailList
= 1
> 1
1105
currentBinding = ListTail (bindingList) Data
1106
tmpTrail = ListHeadRemove (trailList)
1107
Address Extend (pathAddress, tmpTrail)
1108
binding = ListTail Data (binding List)
1109
!binding -> Key && map -> Target key
NO
YES
1110
binding --> Key = map --> Target key
1111
map --> Target key = Null
1112
session
YES
NO
1113
returnList = Prepare Multicast Paths (trailList, map)
Return (multiple)
Return (break)

Fig. 11

Fig. 12


Get Key
edge = binding --> Edge
Edge protocol = edge --> Protocol
1201
Status = edge --> DemuxKey (message, pathAddress, map)
1202
status
1203
remove
postpone
other
binding --> Flags
1 = Binding-Remove
1204
traverse = ListDataNext
postpone++
1205
Return (next binding)
postpone
1206
NO
YES
traverse = ListDataNext
1211
postpone --
traverse = ListDataPrev
1207
saved status
1212
NO
YES
status = saved status
savedstatus = 0
1213
savedStatus = Status
1209
saved status
1208
NO
YES
status = demux
continue
1210
binding -> Flags & Binding Remove
1214
NO
YES
Return (continue)
Return (next binding)

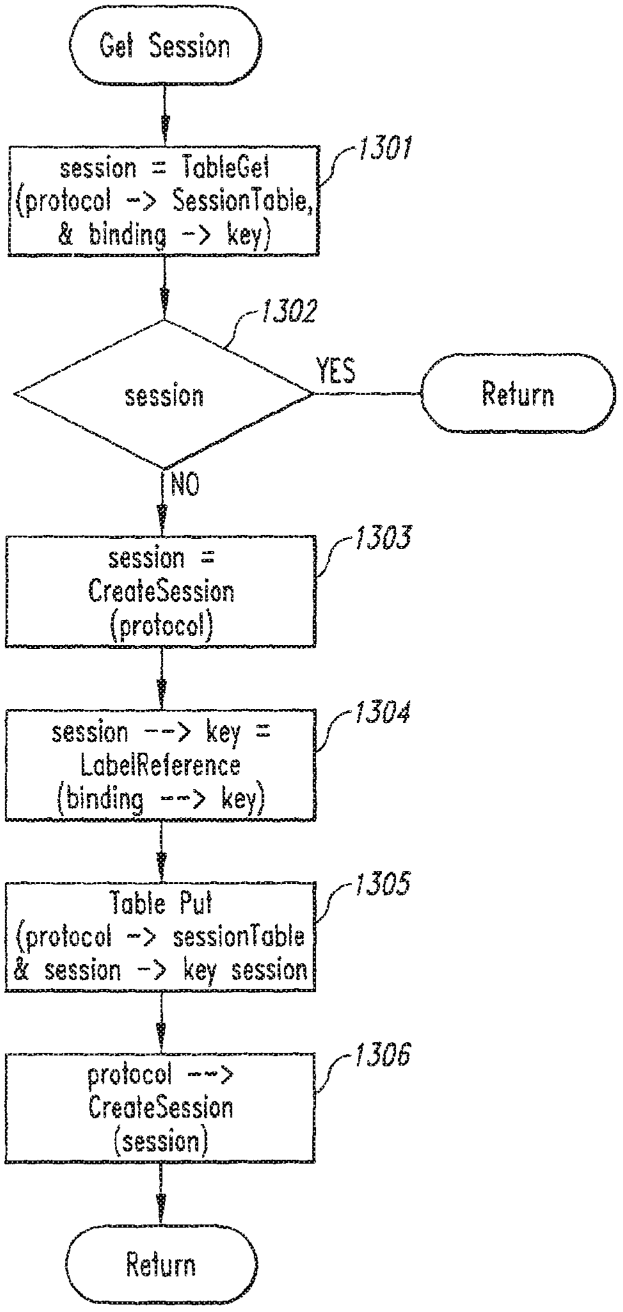

Get Session
session = TableGet (protocol -> SessionTable, & binding -> key)
1301
session
1302
YES
Return
NO
session = CreateSession (protocol)
1303
session --> key = LabelReference (binding --> key)
1304
Table Put (protocol -> sessionTable & session -> key session
1305
protocol --> CreateSession (session)
1306
Return

Fig. 13

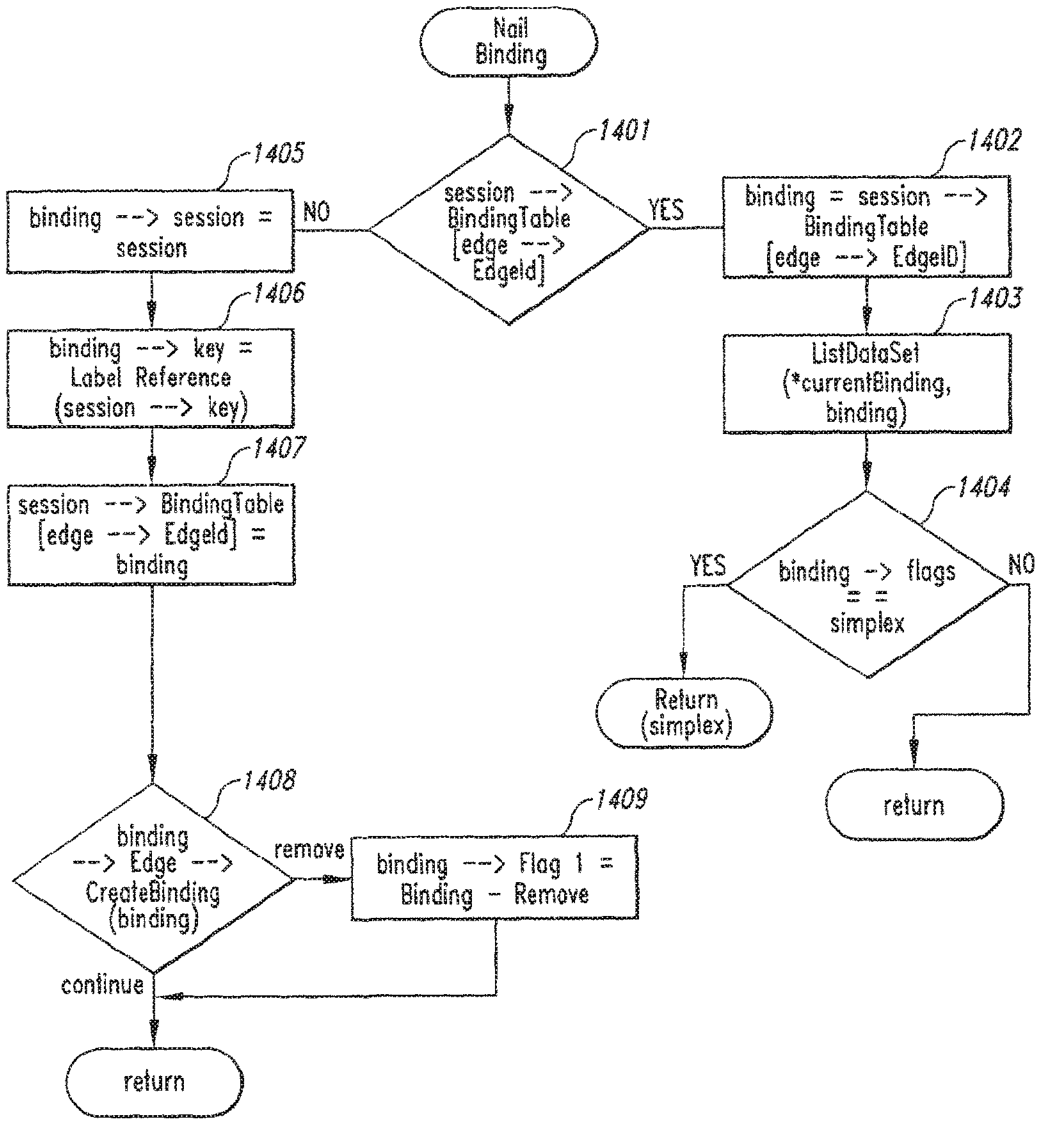

Nail Binding
1401
session --> BindingTable [edge --> EdgeId]
YES
NO
1402
binding = session --> BindingTable [edge --> EdgeID]
1403
ListDataSet (*currentBinding, binding)
1404
binding -> flags = = simplex
YES
NO
Return (simplex)
return
1405
binding --> session = session
1406
binding --> key = Label Reference (session --> key)
1407
session --> BindingTable [edge --> EdgeId] = binding
1408
binding --> Edge --> CreateBinding (binding)
remove
1409
binding --> Flag 1 = Binding - Remove
continue
return

Fig. 14

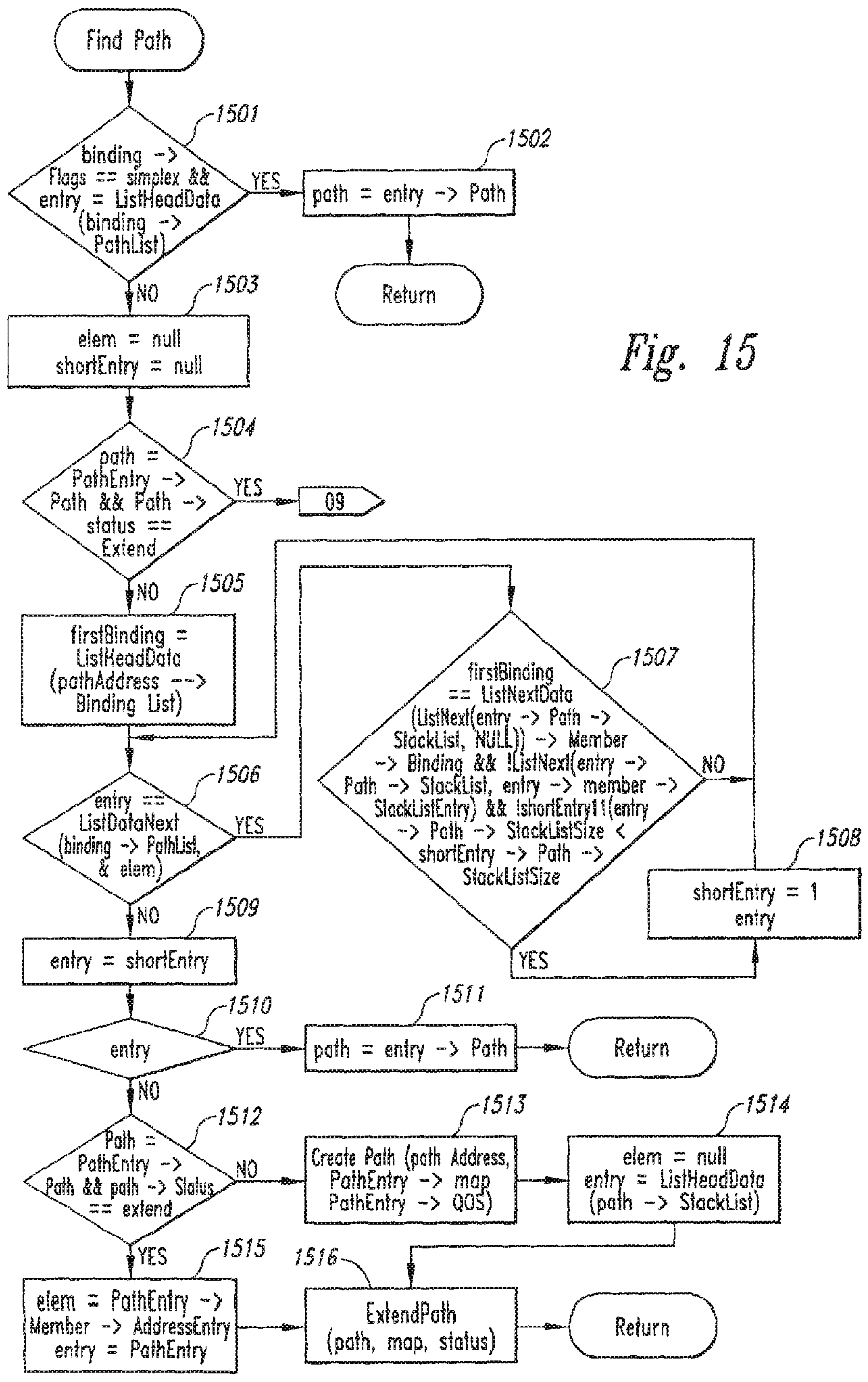

Find Path
1501
binding -> Flags == simplex && entry = ListHeadData (binding -> PathList)
YES
1502
path = entry -> Path
Return
NO
1503
elem = null
shortEntry = null
1504
path = PathEntry -> Path && Path -> status == Extend
YES
09
NO
1505
firstBinding = ListHeadData (pathAddress --> Binding List)
1506
entry == ListDataNext (binding -> PathList, & elem)
YES
1507
firstBinding == ListNextData (ListNext(entry -> Path -> StackList, NULL)) -> Member -> Binding && !ListNext(entry -> Path -> StackList, entry -> member -> StackListEntry) && !shortEntry11(entry -> Path -> StackListSize < shortEntry -> Path -> StackListSize
NO
YES
1508
shortEntry = 1 entry
NO
1509
entry = shortEntry
1510
entry
YES
1511
path = entry -> Path
Return
NO
1512
Path = PathEntry -> Path && path -> Status == extend
NO
1513
Create Path (path Address, PathEntry -> map PathEntry -> QOS)
1514
elem = null
entry = ListHeadData (path -> StackList)
YES
1515
elem = PathEntry -> Member -> AddressEntry
entry = PathEntry
1516
ExtendPath (path, map, status)
Return

Fig. 15

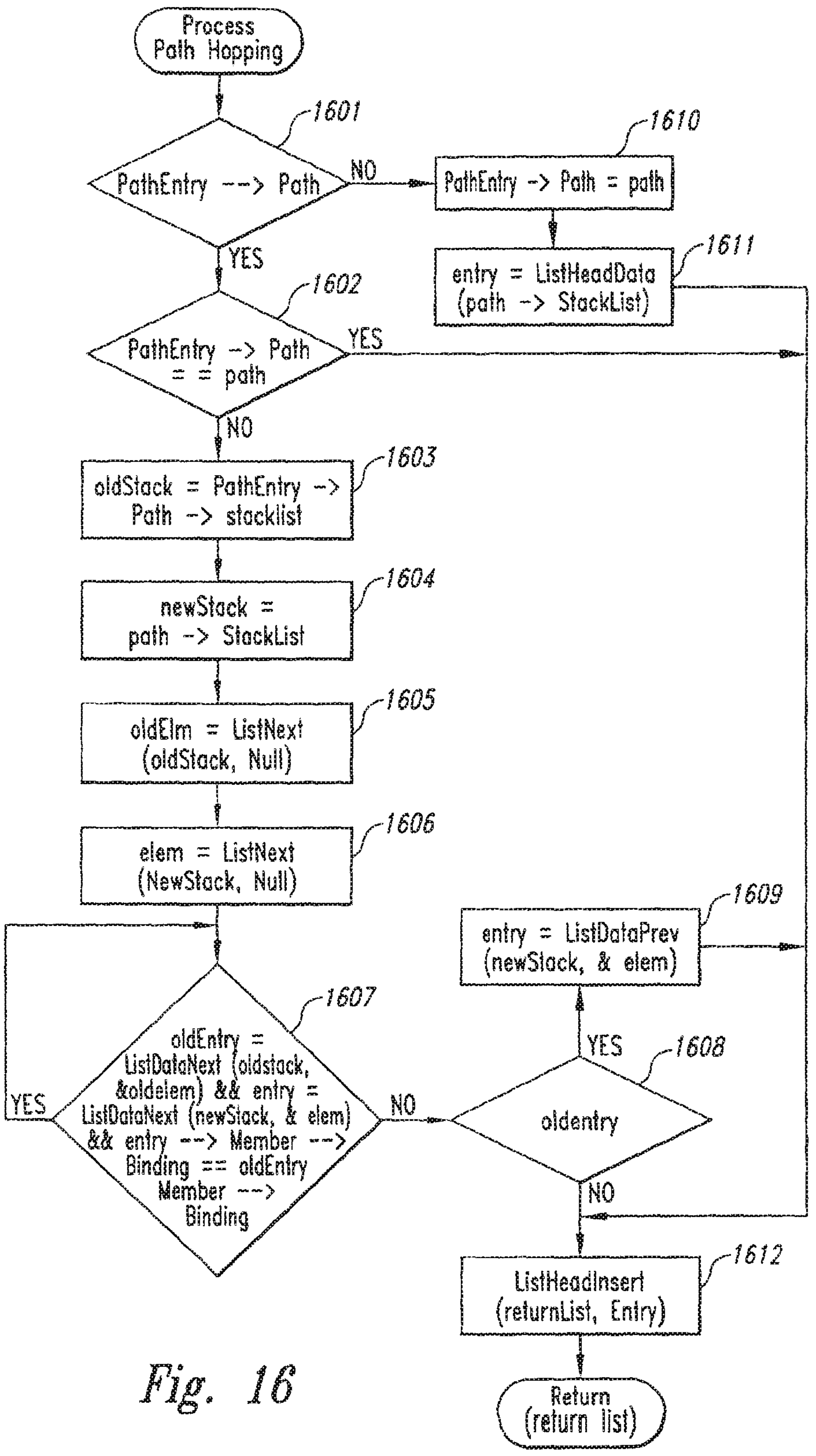

Process Path Hopping
1601
PathEntry --> Path
NO
YES
1610
PathEntry -> Path = path
1611
entry = ListHeadData (path -> StackList)
1602
PathEntry -> Path = = path
YES
NO
1603
oldStack = PathEntry -> Path -> stacklist
1604
newStack = path -> StackList
1605
oldElm = ListNext (oldStack, Null)
1606
elem = ListNext (NewStack, Null)
1607
oldEntry = ListDataNext (oldstack, &oldelem) && entry = ListDataNext (newStack, & elem) && entry --> Member --> Binding == oldEntry Member --> Binding
YES
NO
1608
oldentry
YES
NO
1609
entry = ListDataPrev (newStack, & elem)
1612
ListHeadInsert (returnList, Entry)
Return (return list)

Fig. 16

METHOD AND SYSTEM FOR DATA DEMULTIPLEXING

CROSS REFERENCES TO RELATED APPLICATIONS

The present application is a continuation of U.S. application Ser. No. 15/450,790, filed Mar. 6, 2017 (now U.S. Pat. No. 10,033,839), which is a continuation of U.S. application Ser. No. 15/050,027, filed Feb. 22, 2016 (now U.S. Pat. No. 9,591,104), which is a continuation of U.S. application Ser. No. 14/230,952, filed Mar. 31, 2014 (now U.S. Pat. No. 9,270,790), which is a continuation of U.S. application Ser. No. 13/911,324, filed Jun. 6, 2013 (now U.S. Pat. No. 8,694,683), which is a continuation of U.S. application Ser. No. 13/236,090, filed Sep. 19, 2011 (now abandoned), which is a continuation of U.S. application Ser. No. 10/636,314, filed Aug. 6, 2003 (now U.S. Pat. No. 8,055,786), which is a continuation of U.S. application Ser. No. 09/474,664, filed Dec. 29, 1999 (now U.S. Pat. No. 6,629,163); the disclosures of each of the above-referenced applications are incorporated by reference herein in their entireties.

TECHNICAL FIELD

The present invention relates generally to a computer system for data demultiplexing.

BACKGROUND

Computer systems, which are becoming increasingly pervasive, generate data in a wide variety of formats. The Internet is an example of interconnected computer systems that generate data in many different formats. Indeed, when data is generated on one computer system and is transmitted to another computer system to be displayed, the data may be converted in many different intermediate formats before it is eventually displayed. For example, the generating computer system may initially store the data in a bitmap format. To send the data to another computer system, the computer system may first compress the bitmap data and then encrypt the compressed data. The computer system may then convert that compressed data into a TCP format and then into an IP format. The IP formatted data may be converted into a transmission format, such as an ethernet format. The data in the transmission format is then sent to a receiving computer system. The receiving computer system would need to perform each of these conversions in reverse order to convert the data in the bitmap format. In addition, the receiving computer system may need to convert the bitmap data into a format that is appropriate for rendering on output device.

In order to process data in such a wide variety of formats, both sending and receiving computer systems need to have many conversion routines available to support the various formats. These computer systems typically use predefined configuration information to load the correct combination of conversion routines for processing data. These computer systems also use a process-oriented approach when processing data with these conversion routines. When using a process-oriented approach, a computer system may create a separate process for each conversion that needs to take place. A computer system in certain situations, however, can be expected to receive data and to provide data in many different formats that may not be known until the data is received. The overhead of statically providing each possible series of conversion routines is very high. For example, a

computer system that serves as a central controller for data received within a home would be expected to process data received via telephone lines, cable TV lines, and satellite connections in many different formats. The central controller would be expected to output the data to computer displays, television displays, entertainment centers, speakers, recording devices, and so on in many different formats. Moreover, since the various conversion routines may be developed by different organizations, it may not be easy to identify that the output format of one conversion routine is compatible with the input format of another conversion routine.

It would be desirable to have a technique for dynamically identifying a series of conversion routines for processing data. In addition, it would be desirable to have a technique in which the output format of one conversion routine can be identified as being compatible with the input format of another conversion routine. It would also be desirable to store the identification of a series of conversion routines so that the series can be quickly identified when data is received.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a block diagram illustrating example processing of a message by the conversion system.

FIG. **2** is a block diagram illustrating a sequence of edges.

FIG. **3** is a block diagram illustrating components of the conversion system in one embodiment.

FIG. **4** is a block diagram illustrating example path data structures in one embodiment.

FIG. **5** is a block diagram that illustrates the interrelationship of the data structures of a path.

FIG. **6** is a block diagram that illustrates the interrelationship of the data structures associated with a session.

FIGS. **7** A, **7**B, and **7**C comprise a flow diagram illustrating the processing of the message send routine.

FIG. **8** is a flow diagram of the demux routine.

FIG. **9** is a flow diagram of the initialize demux routine.

FIG. **10** is a flow diagram of the init end routine.

FIG. **11** is a flow diagram of a routine to get the next binding.

FIG. **12** is a flow diagram of the get key routine.

FIG. **13** is a flow diagram of the get session routine.

FIG. **14** is a flow diagram of the nail binding routine.

FIG. **15** is a flow diagram of the find path routine.

FIG. **16** is a flow diagram of the process of path hopping routine.

DETAILED DESCRIPTION

A method and system for converting a message that may contain multiple packets from an source format into a target format. When a packet of a message is received, the conversion system in one embodiment searches for and identifies a sequence of conversion routines (or more generally message handlers) for processing the packets of the message by comparing the input and output formats of the conversion routines. (A message is a collection of data that is related in some way, such as stream of video or audio data or an email message.) The identified sequence of conversion routines is used to convert the message from the source format to the target format using various intermediate formats. The conversion system then queues the packet for processing by the identified sequence of conversion routines. The conversion system stores the identified sequence so that the sequence can be quickly found (without searching) when the next packet in the message is received. When subsequent packets of the message are received, the conversion system identifies the sequence and queues the packets for pressing by the sequence. Because the conversion system receives multiple messages with different source and target formats and identifies a sequence of conversion routines for each message, the conversion systems effectively "demultiplexes" the messages. That is, the conversion system demultiplexes the messages by receiving the message, identifying the sequence of conversion routines, and controlling the processing of each message by the identified sequence. Moreover, since the conversion routines may need to retain state information between the receipt of one packet of a message and the next packet of that message, the conversion system maintains state information as an instance or session of the conversion routine. The conversion system routes all packets for a message through the same session of each conversion routine so that the same state or instance information can be used by all packets of the message. A sequence of sessions of conversion routines is referred to as a "path." In one embodiment, each path has a path thread associated with it for processing of each packet destined for that path.

In one embodiment, the packets of the messages are initially received by "drivers," such as an Ethernet driver. When a driver receives a packet, it forwards the packet to a forwarding component of the conversion system. The forwarding component is responsible for identifying the session of the conversion routine that should next process the packet and invoking that conversion routine. When invoked by a driver, the forwarding component may use a demultiplexing ("demux") component to identify the session of the first conversion routine of the path that is to process the packet and then queues the packet for processing by the path. A path thread is associated with each path. Each path thread is responsible for retrieving packets from the queue of its path and forwarding the packets to the forwarding component. When the forwarding component is invoked by a path thread, it initially invokes the first conversion routine in the path. That conversion routine processes the packet and forwards the processed packet to the forwarding component, which then invokes the second conversion routine in the path. The process of invoking the conversion routines and forwarding the processed packet to the next conversion routine continues until the last conversion routine in the path is invoked. A conversion routine may defer invocation of the forwarding component until it aggregates multiple packets or may invoke the forwarding component multiple times for a packet once for each sub-packet.

The forwarding component identifies the next conversion routine in the path using the demux component and stores that identification so that the forwarding component can quickly identify the conversion routine when subsequent packets of the same message are received. The demux component, searches for the conversion routine and session that is to next process a packet. The demux component then stores the identification of the session and conversion routine as part of a path data structure so that the conversion system does not need to search for the session and conversion routine when requested to demultiplex subsequent packets of the same message. When searching for the next conversion routine, the demux component invokes a label map get component that identifies the next conversion routine. Once the conversion routine is found, the demux component identifies the session associated with that message by, in one embodiment, invoking code associated with the conversion routine. In general, the code of the conversion routine determines what session should be associated with a message. In certain situations, multiple messages may

share the same session. The demux component then extends the path for processing that packet to include that session and conversion routine. The sessions are identified so that each packet is associated with the appropriate state information. The dynamic identification of conversion routines is described in U.S. patent application Ser. No. 11/933,093, filed on Oct. 31, 2007 (now U.S. Pat. No. 7,730,211), entitled "Method and System for Generating a Mapping Between Types of Data," which is hereby incorporated by reference.

FIG. **1** is a block diagram illustrating example processing of a message by the conversion system. The driver **101** receives the packets of the message from a network. The driver performs any appropriate processing of the packet and invokes a message send routine passing the processed packet along with a reference path entry **150**. The message send routine is an embodiment of the forwarding component. A path is represented by a series of path entries, which are represented by triangles. Each member path entry represents a session and conversion routine of the path, and a reference path entry represents the overall path. The passed reference path entry **150** indicates to the message send routine that it is being invoked by a driver. The message send routine invokes the demux routine **102** to search for and identify the path of sessions that is to process the packet. The demux routine may in turn invoke the label map get routine **104** to identify a sequence of conversion routines for processing the packet. In this example, the label map get routine identifies the first three conversion routines, and the demux routine creates the member path entries **151**, **152**, **153** of the path for these conversion routines. Each path entry identifies a session for a conversion routine, and the sequence of path entries **151-155** identifies a path. The message send routine then queues the packet on the queue **149** for the path that is to process the packets of the message. The path thread **105** for the path retrieves the packet from the queue and invokes the message send routine **106** passing the packet and an indication of the path. The message send routine determines that the next session and conversion routine as indicated by path entry **151** has already been found. The message send routine then invokes the instance of the conversion routine for the session. The conversion routine processes the packet and then invokes the message send routine **107**. This processing continues until the message send routine invokes the demux routine **110** after the packet is processed by the conversion routine represented by path entry **153**. The demux routine examines the path and determines that it has no more path entries. The demux routine then invokes the label map get routine **111** to identify the conversion routines for further processing of the packet. When the conversion routines are identified, the demux routine adds path entries **154**, **155** to the path. The messages send routine invokes the conversion routine associated with path entry **154**. Eventually, the conversion routine associated with path entry **155** performs the final processing for the path.

The label map get routine identifies a sequence of "edges" for converting data in one format into another format. Each edge corresponds to a conversion routine for converting data from one format to another. Each edge is part of a "protocol" (or more generally a component) that may include multiple related edges. For example, a protocol may have edges that each convert data in one format into several different formats. Each edge has an input format and an output format. The label map get routine identifies a sequence of edges such that the output format of each edge is compatible with the input format of another edge in the sequence, except for the input format of the first edge in the sequence and the output format of the last edge in the sequence. FIG. **2** is a block diagram illustrating a sequence of edges. Protocol PI includes an edge for converting format D1 to format D2 and an edge for converting format D1 to format D3; protocol P2 includes an edge for converting format D2 to format D5, and so on. A 30 sequence for converting format D1 to format D15 is shown by the curved lines and is defined by the address "P1:I, P2: 1, P3:2, P4:7." When a packet of data in format D I is processed by this sequence, it is converted to format DIS. During the process, the packet of data is sequentially converted to format D2, D5, and D13. The output format of protocol P2, edge 1 (i.e., P2:1) is format D5, but the input format of P3:2 is format D10. The label map get routine uses an aliasing mechanism by which two formats, such as D5 and D10 are identified as being compatible. The use of aliasing allows different names of the same format or compatible formats to be correlated.

FIG. **3** is a block diagram illustrating components of the conversion system in one embodiment. The conversion system **300** can operate on a computer system with a central processing unit **301**, I/O devices **302**, and memory **303**. The **110** devices may include an Internet connection, a connection to various output devices such as a television, and a connection to various input devices such as a television receiver. The media mapping system may be stored as instructions on a computer-readable medium, such as a disk drive, memory, or data transmission medium. The data structures of the media mapping system may also be stored on a computer-readable medium. The conversion system includes drivers **304**, a forwarding component **305**, a demux component **306**, a label map get component **307**, path data structures **308**, conversion routines **309**, and instance data **310**. Each driver receives data in a source format and forwards the data to the forwarding component. The forwarding component identifies the next conversion routine in the path and invokes that conversion routine to process a packet. The forwarding component may invoke the demux component to search for the next conversion routine and add that conversion routine to the path. The demux component may invoke the label map get component to identify the next conversion routine to process the packet. The demux component stores information defining the paths in the path structures. The conversion routines store their state information in the instance data.

FIG. **4** is a block diagram illustrating example path data structures in one embodiment. The demux component identifies a sequence of "edges" for converting data in one format into another format by invoking the label map get component. Each edge corresponds to a conversion routine for converting data from one format to another. As discussed above, each edge is part of a "protocol" that may include multiple related edges. For example, a protocol may have edges that each convert data in one format into several different formats. Each edge has as an input format ("input label") and an output format ("output label"). Each rectangle represents a session **410**, **420**, **430**, **440**, **450** for a protocol. A session corresponds to an instance of a protocol. That is, the session includes the protocol and state information associated with that instance of the protocol. Session **410** corresponds to a session for an Ethernet protocol; session **420** corresponds to a session for an IP protocol; and sessions **430**, **440**, **450** correspond to sessions for a TCP protocol. FIG. **4** illustrates three paths **461**, **462**, **463**. Each path includes edges **411**, **421**, **431**. The paths share the same Ethernet session **410** and IP session **420**, but each path has a unique TCP session **430**, **440**, **450**. Thus, path **461** includes sessions **410**, **420**, and **430**; path **462** includes sessions **410**, **420**, and **440**; and path **463** includes sessions **410**, **420**, and **450**. The conversion system represents each path by a sequence of path entry structures. Each path entry structure is represented by a triangle. Thus, path **461** is represented by path entries **415**, **425**, and **433**. The conversion system represents the path entries of a path by a stack list. Each path also has a queue **471**, **472**, **473** associated with it. Each queue stores the messages that are to be processed by the conversion routines of the edges of the path. Each session includes a binding **412**, **422**, **432**, **442**, **452** that is represented by an oblong shape adjacent to the corresponding edge. A binding for an edge of a session represents those paths that include the edge. The binding **412** indicates that three paths are bound (or "nailed") to edge **411** of the Ethernet session **410**. The conversion system uses a path list to track the paths that are bound to a binding. The path list of binding **412** identifies path entries **413**, **414**, and **415**.

FIG. **5** is a block diagram that illustrates the interrelationship of the data structures of a path. Each path has a corresponding path structure **501** that contains status information and pointers to a message queue structure **502**, a stack list structure **503**, and a path address structure **504**. The status of a path can be extend, continue, or end. Each message handler returns a status for the path. The status of extend means that additional path entries should be added to the path. The status of end means that this path should end at this point and subsequent processing should continue at a new path. The status of continue means that the protocol does not care how the path is handled. In one embodiment, when a path has a status of continue, the system creates a copy of the path and extends the copy. The message queue structure identifies the messages (or packets of a message) that are queued up for processing by the path and identifies the path entry at where the processing should start. The stack list structure contains a list of pointers to the path entry structures **505** that comprise the path. Each path entry structure contains a pointer to the corresponding path data structure, a pointer to a map structure **507**, a pointer to a multiplex list **508**, a pointer to the corresponding path address structure, and a pointer to a member structure **509**. A map structure identifies the output label of the edge of the path entry and optionally a target label and a target key. A target key identifies the session associated with the protocol that converts the packet to the target label. (The terms "media," "label," and "format" are used interchangeably to refer to the output of a protocol.) The multiplex list is used during the demux process to track possible next edges when a path is being identified as having more than one next edge. The member structure indicates that the path entry represents an edge of a path and contains a pointer to a binding structure to which the path entry is associated (or "nailed"), a stack list entry is the position of the path entry within the associated stack list, a path list entry is the position of the path entry within the associated path list of a binding and an address entry is the position of the binding within the associated path address. A path address of a path identifies the bindings to which the path entries are bound. The path address structure contains a URL for the path, the name of the path identified by the address, a pointer to a binding list structure **506**, and the identification of the current binding within the binding list. The URL (e.g., "protocol://tcp(0)/ip (0)/eth(0)") identifies conversion routines (e.g., protocols and edges) of a path in a human-readable format. The URL (universal resource locator) includes a type field (e.g., "protocol") followed by a sequence of items (e.g., "tcp(0)"). The type field specifies the format of the following information in the URL, that specifies that the type field is followed by a sequence of items. Each item identifies a protocol and an edge (e.g., the protocol is "tcp" and the edge is "0"). In one embodiment, the items of a URL may also contain an identifier of state information that is to be used when processing a message. These URLs can be used to illustrate to a user various paths that are available for processing a message. The current binding is the last binding in the path as the path is being built. The binding list structure contains a list of pointers to the binding structures associated with the path. Each binding structure **510** contains a pointer to a session structure, a pointer to an edge structure, a key, a path list structure, and a list of active paths through the binding. The key identifies the state information for a session of a protocol. A path list structure contains pointers to the path entry structures associated with the binding.

FIG. **6** is a block diagram that illustrates the interrelationship of the data structures associated with a session. A session structure **601** contains the context for the session, a pointer to a protocol structure for the session, a pointer to a binding table structure **602** for the bindings associated with the session, and the key. The binding table structure contains a list of pointers to the binding structures **510** for the session. The binding structure is described above with reference to FIG. **5**. The path list structure **603** of the binding structure contains a list of pointers to path entry structures **505**. The path entry structures are described with reference to FIG. **5**.

FIGS. 7A, 7B, and 7C comprise a flow diagram illustrating the processing of the message send routine. The message send routine is passed a message along with the path entry associated with the session that last processed the message. The message send routine invokes the message handler of the next edge in the path or queues the message for processing by a path. The message handler invokes the demux routine to identify the next path entry of the path. When a driver receives a message, it invokes the message send routine passing a reference path entry. The message send routine examines the passed path entry to determine (1) whether multiple paths branch from the path of the passed path entry, (2) whether the passed path entry is a reference with an associated path, or (3) whether the passed path entry is a member with a next path entry. If multiple paths branch from the path of the passed path entry, then the routine recursively invokes the message send routine for each path. If the path entry is a reference with an associated path, then the driver previously invoked the message send routine, which associated a path with the reference path entry, and the routine places the message on the queue for the path. If the passed path entry is a member with a next path entry, then the routine invokes the message handler (i.e., conversion routine of the edge) associated with the next path entry. If the passed path entry is a reference without an associated path or is a member without a next path entry, then the routine invokes the demux routine to identify the next path entry. The routine then recursively invokes the messages send routine passing that next path entry. In decision block **701**, if the passed path entry has a multiplex list, then the path branches off into multiple paths and the routine continues at block **709**, else the routine continues at block **702**. A packet may be processed by several different paths. For example, if a certain message is directed to two different output devices, then the message is processed by two different paths. Also, a message may need to be processed by multiple partial paths when searching for a complete path. In decision block **702**, if the passed path entry is a member, then either the next path entry indicates a nailed binding or the path needs to be extended and the routine continues at block **704**, else the routine continues at block **703**. A nailed binding is a binding (e.g., edge and protocol) is associated with a session. In decision block **703**, the passed path entry is a reference and if the passed path entry has an associated path, then the routine can queue the message for the associated path and the routine continues at block **703**A, else the routine needs to identify a path and the routine continues at block **707**. In block **703**A, the routine sets the entry to the first path entry in the path and continues at block **717**. In block **704**, the routine sets the variable position to the stack list entry of the passed path entry. In decision block **705**, the routine sets the variable next entry to the next path entry in the path. If there is a next entry in the path, then the next session and edge of the protocol have been identified and the routine continues at block **706**, else the routine continues at block **707**. In block **706**, the routine passes the message to the message handler of the edge associated with the next entry and then returns. In block **706**, the routine invokes the demux routine passing the passed message, the address of the passed path entry, and the passed path entry. The demux routine returns a list of candidate paths for processing of the message. In decision block **708**, if at least one candidate path is returned, then the routine continues at block **709**, else the routine returns.

Blocks **709**-**716** illustrate the processing of a list of candidate paths that extend from the passed path entry. In blocks **710**-**716**, the routine loops selecting each candidate path and sending the message to be process by each candidate path. In block **710**, the routine sets the next entry to the first path entry of the next candidate path. In decision block **711**, if all the candidate paths have not yet been processed, then the routine continues at block **712**, else the routine returns. In decision block **712**, if the next entry is equal to the passed path entry, then the path is to be extended and the routine continues at block **705**, else the routine continues at block **713**. The candidate paths include a first path entry that is a reference path entry for new paths or that is the last path entry of a path being extended. In decision block **713**, if the number of candidate paths is greater than one, then the routine continues at block **714**, else the routine continues at block **718**. In decision block **714**, if the passed path entry has a multiplex list associated with it, then the routine continues at block **716**, else the routine continues at block **715**. In block **715**, **11** the routine associates the list of candidate path with the multiplex list of the passed path entry and continues at block **716**. In block **716**, the routine sends the message to the next entry by recursively invoking the message send routine. The routine then loops to block **710** to select the next entry associated with the next candidate path.

Blocks **717**-**718** are performed when the passed path entry is a reference path entry that has a path associated with it. In block **717**, if there is a path associated with the next entry, then the routine continues at block **718**, else the routine returns. In block **718**, the routine queues the message for the path of the next entry and then returns.

FIG. **8** is a flow diagram of the demux routine. This routine is passed the packet (message) that is received, an address structure, and a path entry structure. The demux routine extends a path, creating one if necessary. The routine loops identifying the next binding (edge and protocol) that is to process the message and "nailing" the binding to a session for the message, if not already nailed. After identifying the nailed binding, the routine searches for the shortest path through the nailed binding, creating a path if none exists. In block **801**, the routine invokes the initialize demux routine. In blocks **802**-**810**, the routine loops identifying a path or portion of a path for processing the passed message. In decision block **802**, if there is a current status, which was returned by the demux key routine that was last invoked (e.g., continue, extend, end, or postpone), then the routine continues at block **803**, else the routine continues at block **811**. In block **803**, the routine invokes the get next binding routine. The get next binding routine returns the next binding in the path. The binding is the edge of a protocol. That routine extends the path as appropriate to include the binding. The routine returns a return status of break, binding, or multiple. The return status of binding indicates that the next binding in the path was found by extending the path as appropriate and the routine continues to "nail" the binding to a session as appropriate. The return status of multiple means that multiple trails (e.g., candidate paths) were identified as possible extensions of the path. In a decision block **804**, if the return status is break, then the routine continues at block **811**. If the return status is multiple, then the routine returns. If the return status is binding, then the routine continues at block **805**. In decision block **805**, if the retrieved binding is nailed as indicated by being assigned to a session, then the routine loops to block **802**, else the routine continues at block **806**. In block **806**, the routine invokes the get key routine of the edge associated with the binding. The get key routine creates the key for the session associated with the message. If a key cannot be created until subsequent bindings are processed or because the current binding is to be removed, then the get key routine returns a next binding status, else it returns a continue status. In decision block **807**, if the return status of the get key routine is next binding, then the routine loops to block **802** to get the next binding, else the routine continues at block **808**. In block **808**, the routine invokes the routine get session. The routine get session returns the session associated with the key, creating a new session if necessary. In block **809**, the routine invokes the routine nail binding. The routine nail binding retrieves the binding if one is already nailed to the session. Otherwise, that routine nails the binding to the session. In decision block **810**, if the nail binding routine returns a status of simplex, then the routine continues at block **811** because only one path can use the session, else the routine loops to block **802**. Immediately upon return from the nail binding routine, the routine may invoke a set map routine of the edge passing the session and a map to allow the edge to set its map. In block **811**, the routine invokes the find path routine, which finds the shortest path through the binding list and creates a path if necessary. In block **812**, the routine invokes the process path hopping routine, which determines whether the identified path is part of a different path. Path hopping occurs when, for example, IP fragments are built up along separate paths, but once the fragments are built up they can be processed by the same subsequent path.

FIG. **9** is a flow diagram of the initialize demux routine. This routine is invoked to initialize the local data structures that are used in the demux process and to identify the initial binding. The demux routine finds the shortest path from the initial binding to the final binding. If the current status is demux extend, then the routine is to extend the path of the passed path entry by adding additional path entries. If the current status is demux end, then the demux routine is ending the current path. If the current status is demux continue, then the demux routine is in the process of continuing to extend or in the process of starting a path identified by the passed address. In block **901**, the routine sets the local map structure to the map structure in the passed path entry structure. The map structure identifies the output label, the target label, and the target key. In the block **902**, the routine initializes the local message structure to the passed message structure and initializes the pointers path and address element to null. In block **903**, the routine sets of the variable saved status to 0 and the variable status to demux continue. The variable saved status is used to track the status of the demux process when backtracking to nail a binding whose nail was postponed. In decision block **904**, if the passed path entry is associated with a path, then the routine continues at block **905**, else the routine continues at block **906**. In block **905**, the routine sets the variable status to the status of that path. In block **906**, if the variable status is demux continue, then the routine continues at block **907**. If the variable status is demux end, then the routine continues at block **908**. If the variable status is demux extend, then the routine continues at block **909**. In block **907**, the status is demux continue, and the routine sets the local pointer path address to the passed address and continues at block **911**. In block **908**, the status is demux end, and the routine invokes the init end routine and continues at block **911**. In block **909**, the status is demux extend, and the routine sets the local path address to the address of the path that contains the passed path entry. In block **910**, the routine sets the address element and the current binding of the path address pointed to by the local pointer path address to the address entry of the member structure of the passed path entry. In the block **911**, the routine sets the local variable status to demux continue and sets the local binding list structure to the binding list structure from the local path address structure. In block **912**, the routine sets the local pointer current binding to the address of the current binding pointed to by local pointer path address and sets the local variable postpone to 0. In block **913**, the routine sets the function traverse to the function that retrieves the next data in a list and sets the local pointer session to null. The routine then returns.

FIG. **10** is a flow diagram of the init end routine. If the path is simplex, then the routine creates a new path from where the other one ended, else the routine creates a copy of the path. In block **1001**, if the binding of the passed path entry is simplex (i.e., only one path can be bound to this binding), then the routine continues at block **1002**, else the routine continues at block **1003**. In block **1002**, the routine sets the local pointer path address to point to an address structure that is a copy of the address structure associated with the passed path entry structure with its current binding to the address entry associated with the passed path entry structure, and then returns. In block **1003**, the routine sets the local pointer path address to point to an address structure that contains the URL of the path that contains the passed path entry. In block **1004**, the routine sets the local pointer element to null to initialize the selection of the bindings. In blocks **1005** through **1007**, the routine loops adding all the bindings for the address of the passed path entry that include and are before the passed path entry to the address pointed to by the local path address. In block **1005**, the routine retrieves the next binding from the binding list starting with the first. If there is no such binding, then the routine returns, else the routine continues at block **1006**. In block **1006**, the routine adds the binding to the binding list of the local path address structure and sets the current binding of the local variable path address. In the block **1007**, if the local pointer element is equal to the address entry of the passed path entry, then the routine returns, else the routine loops to block **1005** to select the next binding.

FIG. **11** is a flow diagram of a routine to get the next binding. This routine returns the next binding from the local binding list. If there is no next binding, then the routine invokes the routine label map get to identify the list of edges ("trails") that will map the output label to the target label. If only one trail is identified, then the binding list of path address is extended by the edges of the trail. If multiple trails are identified, then a path is created for each trail and the routine returns so that the demux process can be invoked for each created path. In block **1101**, the routine sets the local pointer binding to point to the next or previous (as indicated by the traverse function) binding in the local binding list. In block **1102**, if a binding was found, then the routine returns an indication that a binding was found, else the routine continues at block **1103**. In block **1103**, the routine invokes the label map get function passing the output label and target label of the local map structure. The label map get function returns a trail list. A trail is a list of edges from the output label to the target label. In decision block **1104**, if the size of the trail list is one, then the routine continues at block **1105**, else the routine continues at block **1112**. In blocks **1105-1111**, the routine extends the binding list by adding a binding data structure for each edge in the trail. The routine then sets the local binding to the last binding in the binding list. In block **1108**, the routine sets the local pointer current binding to point to the last binding in the local binding list. In block **1106**, the routine sets the local variable temp trail to the trail in the trail list. In block **1107**, the routine extends the binding list by temp trail by adding a binding for each edge in the trail. These bindings are not yet nailed. In block **1108**, the routine sets the local binding to point to the last binding in the local binding list. In decision block **1109**, if the local binding does not have a key for a session and the local map has a target key for a session, then the routine sets the key for the binding to the target key of the local map and continues at block **1110**, else the routine loops to block **1101** to retrieve the next binding in path. In block **1110**, the routine sets the key of the local binding to the target key of the local map. In block **1111**, the routine sets the target key of the local map to null and then loop to block **1101** to return the next binding. In decision block **1112**, if the local session is set, then the demultiplexing is already in progress and the routine returns a break status. In block **1113**, the routine invokes a prepare multicast paths routine to prepare a path entry for each trail in the trail list. The routine then returns a multiple status.

FIG. **12** is a flow diagram of the get key routine. The get key routine invokes an edge's demux key routine to retrieve a key for the session associated with the message. The key identifies the session of a protocol. The demux key routine creates the appropriate key for the message. The demux key routine returns a status of remove, postpone, or other. The status of remove indicates that the current binding should be removed from the path. The status of postpone indicates that the demux key routine cannot create the key because it needs information provided by subsequent protocols in the path. For example, a TCP session is defined by a combination of a remote and local port address and an IP address. Thus, the TCP protocol postpones the creating of a key until the IP protocol identifies the IP address. The get key routine returns a next binding status to continue at the next binding in the path. Otherwise, the routine returns a continue status. In block **1201**, the routine sets the local edge to the edge of the local binding (current binding) and sets the local protocol to the protocol of the local edge. In block **1202**, the routine invokes the demux key routine of the local edge passing the local message, local path address, and local map. The demux key routine sets the key in the local binding. In decision block **1203**, if the demux key routine returns a status of remove, then the routine continues at block **1204**. If the demux key routine returns a status of postpone, then the routine continues at block **1205**, else the routine continues at block **1206**. In block **1204**, the routine sets the flag of the

local binding to indicate that the binding is to be removed and continues at block **1206**. In block **1205**, the routine sets the variable traverse to the function to list the next data, increments the variable postpone, and then returns a next binding status. In blocks **1206-1214**, the routine processes the postponing of the creating of a key. In blocks **1207-1210**, if the creating of a key has been postponed, then the routine indicates to backtrack on the path, save the demux status, and set the demux status to demux continue. In blocks **1211-1213**, if the creating of a key has not been postponed, then the routine indicates to continue forward in the path and to restore any saved demux status. The save demux status is the status associated by the binding where the backtrack started. In decision block **1206**, if the variable postpone is set, then the routine continues at block **1207**, else the routine continues at block **1211**. In block **1207**, the routine decrements the variable postpone and sets the variable traverse to the list previous data function. In decision block **1208**, if the variable saved status is set, then the routine continues at block **1210**, else the routine continues at block **1209**. The variable saved status contains the status of the demux process when the demux process started to backtrack. In block **1209**, the routine sets the variable saved status to the variable status. In block **1210**, the routine sets the variable status to demux continue and continues at block **1214**. In block **1211**, the routine sets the variable traverse to the list next data function. In decision block **1212**, if the variable saved status in set, then the routine continues at block **1213**, else the routine continues at block **1214**. In block **1213**, the routine sets the variable status to the variable saved status and sets the variable saved status to 0. In decision block **1214**, if the local binding indicates that it is to be removed, then the routine returns a next binding status, else the routine returns a continue status.

FIG. **13** is a flow diagram of the get session routine. This routine retrieves the session data structure, creating a data structure session if necessary, for the key indicated by the binding. In block **1301**, the routine retrieves the session from the session table of the local protocol indicated by the key of the local binding. Each protocol maintains a mapping from each key to the session associated with the key. In decision block **1302**, if there is no session, then the routine continues at block **1303**, else the routine returns. In block **1303**, the routine creates a session for the local protocol. In block **1304**, the routine initializes the key for the local session based on the key of the local binding. In block **1305**, the routine puts the session into the session table of the local protocol. In block **1306**, the routine invokes the create session function of the protocol to allow the protocol to initialize its context and then returns.

FIG. **14** is a flow diagram of the nail binding routine. This routine determines whether a binding is already associated with ("nailed to") the session. If so, the routine returns that binding. If not, the routine associates the binding with the session. The routine returns a status of simplex to indicate that only one path can extend through the nailed binding. In decision block **1401**, if the binding table of the session contains an entry for the edge, then the routine continues at block **1402**, else the routine continues at block **1405**. In block **1402**, the routine sets the binding to the entry from the binding table of the local session for the edge. In block **1403**, the routine sets the current binding to point to the binding from the session. In block **1404**, if the binding is simplex, then the routine returns a simplex status, else the routine returns. Blocks **1405** through **1410** are performed when there is no binding in the session for the edge. In block **1405**, the routine sets the session of the binding to the variable session. In block **1406**, the routine sets the key of the binding to the key from the session. In block **1407**, the routine sets the entry for the edge in the binding table of the local session to the binding. In block **1408**, the routine invokes the create binding function of the edge of the binding passing the binding so the edge can initialize the binding. If that function returns a status of remove, the routine continues at block **1409**. In block **1409**, the routine sets the binding to be removed and then returns.

FIG. **15** is a flow diagram of the find path routine. The find path routine identifies the shortest path through the binding list. If no such path exists, then the routine extends a path to include the binding list. In decision block **1501**, if the binding is simplex and a path already goes through this binding (returned as an entry), then the routine continues at block **1502**, else the routine continues at block **1503**. In block **1502**, the routine sets the path to the path of the entry and returns. In block **1503**, the routine initializes the pointers element and short entry to null. In block **1504**, the routine sets the path to the path of the passed path entry. If the local path is not null and its status is demux extend, then the routine continues at block **1509**, else the routine continues at block **1505**. In blocks **1505-1508**, the routine loops identifying the shortest path through the bindings in the binding list. The routine loops selecting each path through the binding. The selected path is eligible if it starts at the first binding in the binding list and the path ends at the binding. The routine loops setting the short entry to the shortest eligible path found so far. In block **1505**, the routine sets the variable first binding to the first binding in the binding list of the path address. In block **1506**, the routine selects the next path (entry) in the path list of the binding starting with the first. If a path is selected (indicating that there are more paths in the binding), then the routine continues at block **1507**, else the routine continues at block **1509**. In block **1507**, the routine determines whether the selected path starts at the first binding in the binding list, whether the selected path ends at the last binding in the binding list, and whether the number of path entries in the selected path is less than the number of path entries in the shortest path selected so far. If these conditions are all satisfied, then the routine continues at block **1508**, else the routine loops to block **1506** to select the next path (entry). In block **1508**, the routine sets the shortest path (short entry) to the selected path and loops to block **1506** to select the next path through the binding. In block **1509**, the routine sets the selected path (entry) to the shortest path. In decision block **1510**, if a path has been found, then the routine continues at block **1511**, else the routine continues at block **1512**. In block **1511**, the routine sets the path to the path of the selected path entry and returns. Blocks **1512-1516** are performed when no paths have been found. In block **1512**, the routine sets the path to the path of the passed path entry. If the passed path entry has a path and its status is demux extend, then the routine continues at block **1515**, else the routine continues at block **1513**. In block **1513**, the routine creates a path for the path address. In block **1514**, the routine sets the variable element to null and sets the path entry to the first element in the stack list of the path. In block **1515**, the routine sets the variable element to be address entry of the member of the passed path entry and sets the path entry to the passed path entry. In block **1516**, the routine invokes the extend path routine to extend the path and then returns. The extend path routine creates a path through the bindings of the binding list and sets the path status to the current demux status.

FIG. **16** is a flow diagram of the process of path hopping routine. Path hopping occurs when the path through the binding list is not the same path as that of the passed path entry. In decision block **1601**, if the path of the passed path entry is set, then the routine continues at block **1602**, else the routine continues at block **1609**. In decision block **1602**, if the path of the passed path entry is equal to the local path, then the routine continues at **1612**, else path hopping is occurring and the routine continues at block **1603**. In blocks **1603-1607**, the routine loops positioning pointers at the first path entries of the paths that are not at the same binding. In block **1603**, the routine sets the variable old stack to the stack list of the path of the passed path entry. In block **1604**, the routine sets the variable new stack to the stack list of the local path. In block **1605**, the routine sets the variable old element to the next element in the old stack. In block **1606**, the routine sets the variable element to the next element in the new stack. In decision block **1607**, the routine loops until the path entry that is not in the same binding is located. In decision block **1608**, if the variable old entry is set, then the routine is not at the end of the hopped from path and the routine continues at block **1609**, else routine continues at block **1612**. In block **1609**, the routine sets the variable entry to the previous entry in the hopped-to path. In block **1610**, the routine sets the path of the passed path entry to the local path. In block **1611**, the routine sets the local entry to the first path entry of the stack list of the local path. In block **1612**, the routine inserts an entry into return list and then returns.

Although the conversion system has been described in terms of various embodiments, the invention is not limited to these embodiments. Modification within the spirit of the invention will be apparent to those skilled in the art. For example, a conversion routine may be used for routing a message and may perform no conversion of the message. Also, a reference to a single copy of the message can be passed to each conversion routine or demux key routine. These routines can advance the reference past the header information for the protocol so that the reference is positioned at the next header. After the demux process, the reference can be reset to point to the first header for processing by the conversion routines in sequence. The scope of the invention is defined by the claims that follow.

What is claimed is:

1. A method, comprising:

receiving, at a computing device, a packet of a message;

determining, by the computing device, a key value for the packet, wherein the key value is determined based on one or more headers in the packet;

using, by the computing device, the key value to determine whether the computing device is currently storing a previously created path for the key value;

in response to determining that no path is currently stored for the key value, the computing device:

identifying, using the key value, one or more routines for processing the packet, including a routine that is used to execute a Transmission Control Protocol (TCP) to convert packets having a TCP format into a different format;

creating a path using the identified one or more routines; and

processing the packet using the created path.

2. The method of claim **1**, wherein the created path stores state information for at least one of the identified one or more routines.

3. The method of claim **1**, wherein the created path stores state information for each of the identified one or more routines.

4. The method of claim **1**, wherein the created path specifies an ordering in which the identified one or more routines are to be performed to process the packet.

5. The method of claim **4**, wherein the ordering specifies that an application layer protocol is to be performed subsequent to the TCP.

6. The method of claim **5**, wherein the application layer protocol is HTTP, and wherein the different format is HTTP.

7. The method of claim **4**, wherein the ordering specifies that a first execution of the TCP is to be followed by execution of an application layer protocol, which is to be followed by a second execution of the TCP.

8. The method of claim **7**, wherein the first execution of the TCP receives information from a network and the second execution of the TCP sends information via the network.

9. The method of claim **4**, wherein the ordering specifies that the TCP is an initial one of the one or more routines.

10. The method of claim **4**, wherein the ordering specifies that the TCP is to be performed after performing an Ethernet protocol.

11. The method of claim **1**, further comprising:
- receiving, at the computing device, a subsequent packet of the message;
- determining, by the computing device based on the subsequent packet, the key value;
- using, by the computing device, the key value to identify the created path for the message; and
- processing, by the computing device, the subsequent packet using the path.

12. The method of claim **11**, wherein processing the subsequent packet includes:
- queuing the subsequent packet for one or more routines specified in the path; and
- performing the one or more routines according to an ordering specified by the path, wherein performing at least one of the routines includes accessing state information stored in the path.

13. The method of claim **11**, wherein packets of the message are all associated with a single TCP session.

14. The method of claim **1**, wherein the key value includes an IP address and one or more port addresses.

15. A method, comprising:
- receiving, at a computing device, a packet of a message;
- determining, by the computing device, a key value for the packet, wherein the key value is determined based on one or more headers in the packet;
- using, by the computing device, the key value to determine whether the computing device is currently storing a previously created path for the key value;
- in response to determining that no path is currently stored for the key value, the computing device:
  - identifying, using the key value, one or more routines for processing the packet, including a routine that is used to execute a User Datagram Protocol (UDP) to convert packets having a UDP format into a different format;
  - creating a path using the identified one or more routines; and
  - processing the packet using the created path.

16. An apparatus, comprising:
- one or more memories storing program instructions executable by the apparatus to:
  - receive, from a network, a packet of a message;
  - determine a key value for the packet, wherein the key value is determined based on one or more headers in the packet;
  - use the key value to determine whether the apparatus is currently storing a path for the key value, wherein one or more routines are specified in the path for processing packets of the message;
  - in response to determining that no path is currently stored for the key value:
    - identify, using the key value, one or more routines for processing the packet, including a particular routine that is used to execute a Transmission Control Protocol (TCP) to convert packets having a TCP format into a different format;
    - create a path using the identified one or more routines;
    - process the packet using the created path; and
    - store the path for use in processing subsequent packets in the message; and
  - in response to determining that a path is currently stored for the key value;
    - process the packet using the stored path.

17. The apparatus of claim **16**, wherein the apparatus is configured to process the packet by queuing the packet for the one or more routines identified in the path.

18. The apparatus of claim **16**, wherein the different format is an application layer format.

19. The apparatus of claim **16**, wherein the particular routine is executable to utilize state information stored within the path.

20. The apparatus of claim **16**, wherein the path stores state information for at least some of the one or more routines.

* * * * *