

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent No.:	6,629,163	Group Art Unit:	To be assigned
Inventors:	Edward Balassanian	Examiner:	To be assigned
Issued:	Sept. 30, 2003	Attorney Docket No.:	159291-0025(163)
Serial No.:	09/474,664	Reexam Control No.:	To be assigned
Title:	METHOD AND SYSTEM FOR DEMULTIPLEXING A FIRST SEQUENCE OF PACKET COMPONENTS TO IDENTIFY SPECIFIC COMPONENTS WHEREIN SUBSEQUENT COMPONENTS ARE PROCESSED WITHOUT RE-IDENTIFYING COMPONENTS	Reexam Filing Date:	To be assigned

REQUEST FOR *INTER PARTES* REEXAMINATION

Mail Stop *Inter Partes* Reexam

Attn: Central Reexamination Unit
Commissioner for Patents
P.O. Box 1450
Alexandria, Virginia 22313-1450

Sir or Madam:

Juniper Networks, Inc. (hereinafter "Requester") respectfully requests *inter partes* reexamination of U.S. Patent No. 6,629,163 ("the '163 patent") entitled "Method and System for Demultiplexing a First Sequence of Packet Components to Identify Specific Components Wherein Subsequent Components are Processed Without Re-Identifying Components." This Request is made pursuant to 35 U.S.C. §§ 311-316 and 37 C.F.R. §§ 1.906, 1.913 and 1.915. The '163 patent was filed on December 29, 1999 and issued on September 30, 2003. The patent has not yet expired. As a result of *ex parte* reexamination, an Ex Parte Reexamination Certificate (7567th) issued for the '163 patent

on June 22, 2010. Implicit Networks, Inc. (“Implicit”) has alleged that it is the current assignee of the ‘163 patent. A copy of the ‘163 patent, in the format specified by 37 C.F.R. § 1.915(b)(5), is attached as Exhibit 1. The reexamination certificate is attached as Exhibit 2.

This Request for *Inter Partes* Reexamination (“Request”) is being served on the correspondent of record for the ‘163 patent (Newman Du Wors LLP, 1201 Third Avenue, Suite 1600, Seattle, WA 98101) and on counsel for Implicit (Hosie Rice LLP, Transamerica Pyramid, 34th Floor, 600 Montgomery Street, San Francisco, CA 94111). This Request is also accompanied by the required fee as set forth in 37 C.F.R. § 1.20(c)(2) and the certificate required by 37 C.F.R. § 1.915(b)(6).

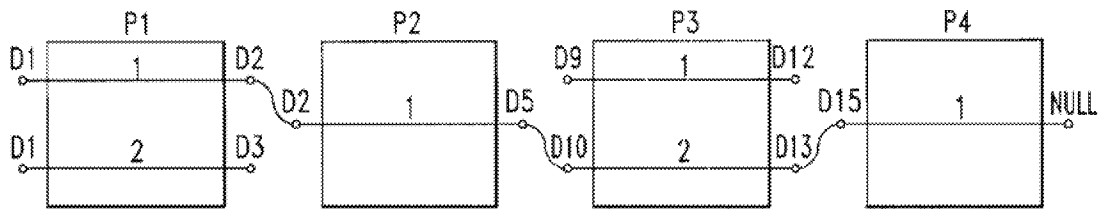
For the convenience of the Examiner, following is a table of contents for this Request:

Major Section	Page
I. INTRODUCTION	3
II. DISCLOSURE OF CONCURRENT PROCEEDINGS	9
III. CLAIMS FOR WHICH REEXAMINATION IS REQUESTED AND CITATION OF PRIOR ART	10
IV. CLAIM CONSTRUCTION ADMISSIONS OF THE PATENT OWNER	18
V. PERTINENCE AND MANNER OF APPLYING THE PRIOR ART	24
VI. CERTIFICATION PURSUANT TO 37 C.F.R. § 1.915(b)(7)	272
VII. IDENTIFICATION OF REAL PARTY IN INTEREST PURSUANT TO 37 C.F.R. § 1.915(b)(8)	272
VIII. CONCLUSION	272

I. INTRODUCTION

The PTO should grant this Request and initiate *inter partes* reexamination proceedings for the ‘163 patent in light of the invalidating prior art presented herein. Virtually all of the art cited in this Request has never before been considered in connection with the ‘163 patent claims, and the art clearly discloses every element of the claims to be reexamined—including those elements that the patentee previously alleged during prosecution to be distinguishing features over the prior art. Given the clear teachings of this new prior art as explained below, this Request readily satisfies the threshold requirement of presenting a “reasonable likelihood that the requester would prevail” with respect to one or more of the challenged claims. 35 U.S.C. 312.

The ‘163 patent describes itself as relating “generally to a computer system for data demultiplexing.” Ex. 1 at 1:11-12, 2:57-64. As explained in the background section of the patent, contemporary computer systems “generate data in a wide variety of formats,” including bitmap, encryption, and compression formats, and formats used for packet-based communications such as TCP and IP. *Id.* at 1:24-29. To facilitate processing of communications in this multi-format environment, the patent proposes a “method and system for converting a message that may contain multiple packets from [a] source format into a target format.” *Id.* at 2:38-40. The packet processing method as claimed employed a “sequence” of components, such that a format conversion could be performed by using a plurality of components taking a message through “various intermediate formats” before reaching the final, target format. *Id.* at 2:47-49. An illustration of such a conversion (from format D1 to D15) is illustrated in Figure 2 of the ‘163 patent:



During the original prosecution and prior *ex parte* reexamination proceedings for the '163 patent, the patentee emphasized a few specific features of its purported invention in an attempt to distinguish prior art cited against it. The original claims as filed in 2003 described a method in which (1) a *packet* of a *message* was *received*, (2) a *component* for *processing* the packet was *identified*, and then (3) certain steps relevant to packet processing were performed involving “*state information*.” In response to an initial office action rejecting all of the original claims, the patentee cancelled those claims and proposed a new set of claims adding language to the effect that the identification of a *sequence of components* for processing must be *stored*, “so that the sequence *does not need to be re-identified* for *subsequent packets* of the message.” In other words, an identification of components was to take place only for the first packet of a given message; that identification was then to be stored and made available for subsequent packets in the message, which could then essentially follow the lead of the first packet through the sequence of components already identified.

The examiner issued a notice of allowance for the claims as thus amended, stating that this new limitation—processing of subsequent packets “without re-identifying” a new sequential order of components—was not taught or suggested in the prior art of record. Indeed, the examiner underscored the importance of the limitation with an examiner’s amendment to the patent title which included the words: “*Wherein Subsequent Components are Processed Without Re-Identifying Components*.”

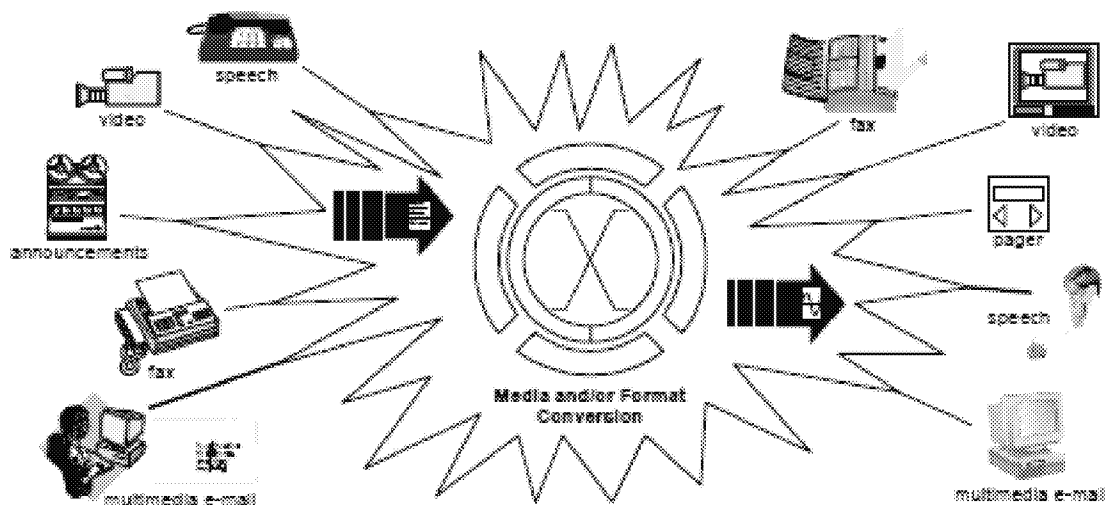
Years later, the PTO initiated *ex parte* reexamination proceedings for the '163 patent on the request of a third party that had been accused of infringing the patent.¹ During those proceedings, the patentee offered a new purported point of distinction in an attempt to overcome the primary piece of prior art under consideration in the reexamination—a paper called the “Mosberger” reference. Specifically, the patentee argued that “[t]he '163 invention is about a system that, upon receipt of first message packet, *dynamically selects* a sequence of components to create a path for processing the message.” Ex. 35-I [Examiner Interview PowerPoint]. In other words, there is a specific, sequential “order to [the] claims – *first, packet is received, and then, component sequence is identified* based on packet.” *Id.* The patentee pointed to language from the specification suggesting the importance of a “dynamic” approach in avoiding the “overhead” that would otherwise be involved in calculating “each possible series of conversion routines” in advance. Ex. 1 at 1:38-66. The patentee alleged that Mosberger, by contrast, performed its identification of sequences *before* the first packet was received, and therefore did not disclose the type of *dynamic* identification contemplated by the claims.

After multiple rejections, the patentee was ultimately forced to amend its claims (though purportedly only to “clarify” their original intent) to expressly include the step of “*dynamically* identifying a *non-predefined* sequence of components.” The examiners in the reexamination unit subsequently issued a notice of allowance for these claims as amended. The allowance was expressly based on the patentee’s argument that “Mosberger does not dynamically identify sequences.”

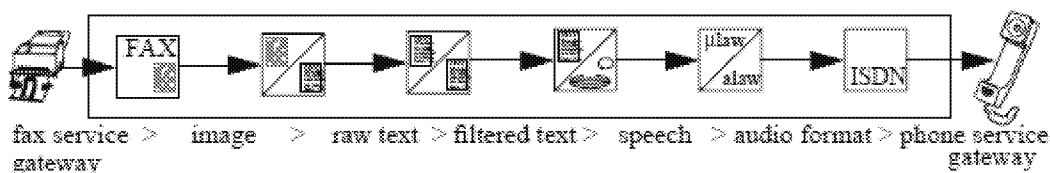
¹ The litigation matter settled before conclusion of the *ex parte* reexamination proceedings.

The new prior art now presented in this Request plainly discloses the very elements of the claimed invention that were supposedly found lacking during prior prosecution of the '163 patent.

For example, a technical paper presented at an international telecommunications conference in 1996 (“Pfeifer96”) demonstrates that researchers had already discovered how to perform *dynamic* conversion from a source format into a target format using a wide variety of formats:



Ex. A02 at 118. To do this, Pfeifer96 teaches the use of what it calls a “*dynamically generated converter chain*”—an approach indistinguishable from that claimed in the '163 patent (compare the following Fig. 6 of Pfeifer96 to Fig. 2 of the '163 patent):

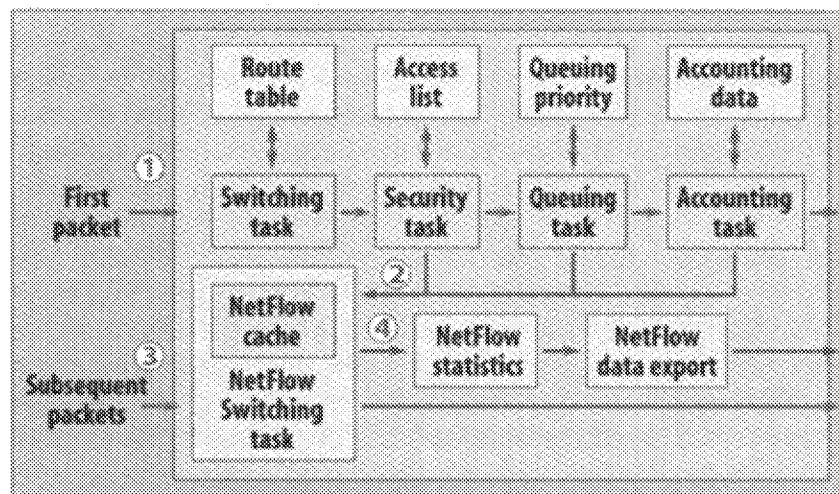


Id. at 111 (Figure 6: “Converter chain: fax reception, conversion to text and speech, telephone delivery”); *see also id.* at 125 (characterizing “converter chain” as “dynamically generated”). This reference was published *over three years* before the

patentee had even filed the application that became the '163 patent. Pfeifer96 fully anticipates and renders obvious every element of claims 1, 15, and 35 of the '163 patent, both on its own and in combinations with other references as set forth in this Request.

And Pfeifer96 is hardly the only example of invalidating prior art dating from years before the critical date of the '163 patent. Cisco Systems was also actively involved in this technological space in 1996, when a pair of Cisco engineers filed an application that ultimately issued as a patent ("Kerr"). The Kerr patent teaches how network administrators can flexibly configure systems with the use of a technology called "flows," in which the first packet of a message goes through several functions task by task and then "caches" the information for high-speed use by subsequent packets. This functionality was incorporated into actual Cisco products under the name "NetFlow," as elaborated in the following article excerpt from a 1997 trade publication:

Cisco streamlines routing, management



With Cisco's NetFlow Switching, the first packet of traffic goes through several functions task by task ①. The NetFlow cache learns about the flow ② and carries out those tasks at high speed for subsequent packets ③. The information gathered about the flow can then be passed on for network management and planning ④.

SOURCE: CISCO SYSTEMS

Ex. 16 [InfoWorld Article]. The Kerr technology as embodied in NetFlow is still part of Cisco's product line to this day.² Kerr fully anticipates and renders obvious every element of claims 1, 15, and 35 of the '163 patent, both on its own and in combinations with other references as set forth in this Request.

This Request contains other invalidating references and combinations of references. For example, a 1998 article ("Decasper98") presents its own solution to the "increasingly rapid pace" with which "[n]ew network protocols . . . are being deployed on the Internet," by proposing an architecture with "code modules, called *plugins*, to be *dynamically added and configured at runtime*." Ex. 25 [Decasper98] at 229. As with Kerr, the "information gathered by processing the *first packet*" is stored in a "cache," from which "*[s]ubsequent packets*" can obtain it "quickly and efficiently." *Id.* at 231.

Finally, although this Request presents numerous prior art references teaching the supposed shortcomings of the Mosberger reference cited in the prior *ex parte* reexamination, it also explains how Mosberger is not nearly so limited as the patentee argued to the PTO during those proceedings. Mosberger itself states that it would be "*straight-forward to add a dynamic module-loading facility*." Ex. 31 [Mosberger] at 71. Thus viewed for the first time in this new light, Mosberger also anticipates and renders obvious the '163 patent claims by itself or in combination with other references.

In summary, for these reasons and as detailed below, there is a reasonable—and indeed compelling—likelihood that Requester will prevail on the proposed claim rejections presented herein. Accordingly, this Request should be granted as to at least

² See <http://www.cisco.com/en/US/products/ps6601/products_ios_protocol_group_home.html>.

claims 1, 15, and 35 of the '163 patent, and a certificate under 35 U.S.C. § 316(a) ultimately issued cancelling all of these claims.

II. DISCLOSURE OF CONCURRENT PROCEEDINGS

Implicit has asserted the '163 patent against Requester in a District Court action styled *Implicit Networks, Inc. v. Juniper Networks, Inc.* (N.D. Cal. Civ. No. Civ. No. 3:10-cv-04234-SI). In the District Court action, Implicit alleges that it is the owner of the '163 patent by assignment. Implicit alleges that claims 1, 15, and 35 of the '163 patent are infringed by Requester's products. For example, in its first amended complaint against Requester, Implicit describes the allegedly infringing functionality as follows:

37. Junos OS dynamically identifies a sequence of actions to be performed on a data packet flow on the basis of the first packet. The sequence of actions so identified is applied to all the subsequent packets of the flow. The actions to be performed are determined using policies maintained by the system. Junos OS inspects data packets, analyzes them against the various policies and performs the appropriate actions as dictated by the applicable policies. Junos OS performs de-multiplexing of data packets by reassembling datagrams fragmented over multiple packets.

38. Whenever a data packet transits Juniper networking equipment running the Junos OS, Junos OS performs a flow lookup to see if the packet belongs to an already established session. If the packet does not belong to an existing session, a new session is created with the packet as the first packet of the session. The system then analyzes the first packet to determine the various actions to be performed on all the data packets of that session. The sequence of actions determined on the basis of the first packet forms a fast processing path. All subsequent packets of the session are then processed through the fast processing path.

Ex. 36-A [Complaint] at 10; *see also* Exs. 36-B – 36-D [Infringement Contentions].

To date, the District Court has not construed any term of the '163 patent, although the parties have briefed claim construction and a Markman hearing was held on January 18-19, 2012.

III. CLAIMS FOR WHICH REEXAMINATION IS REQUESTED AND CITATION OF PRIOR ART

Reexamination of claims 1, 15, and 35 of the '163 patent is requested under 35 U.S.C. §§ 311-316 and 37 C.F.R. §§ 1.906, 1.913 and 1.915 based on the following references:

Prior Art Reference	Prior Art Date	Exhibit
Article entitled "Generic Conversion of Communication Media for Supporting Personal Mobility" by Tom Pfeifer and Radu Popescu-Zeletin ("Pfeifer96")	November 27, 1996	Ex. 3
Color version of Pfeifer96 ("Pfeifer96a")	November 27, 1996	Ex. 3-B
Specification entitled "ISDN Primary Rate User-Network Interface Specification" from Northern Telecom ("ISDN98")	August 1998	Ex. 4
Book entitled "The Data Compression Book" by Mark Nelson and Jean-Loup Gailly ("Nelson")	November 6, 1995	Ex. 5
Book entitled "Superdistribution: Objects as Property on the Electronic Frontier" by Brad Cox ("Cox")	June 4, 1996	Ex. 6
Thesis entitled "Job and Stream Control in Heterogeneous Hardware and Software Architectures" by Stefan Franz ("Franz98")	April 22, 1998	Ex. 7
Thesis entitled "Dynamic Configuration Management of the Equipment in Distributed Communication Environments" by Sven van der Meer ("Meer96")	October 6, 1996	Ex. 8

Prior Art Reference	Prior Art Date	Exhibit
Specification entitled RFC 793: "Transmission Control Protocol" by Information Sciences Institute ("RFC 793")	September 1981	Ex. 9
Book entitled "Principles of Information Systems Analysis and Design" by Harlan D. Mills, Richard C. Linger, and Alan R. Hevner ("Mills")	1986	Ex. 10
Thesis entitled "Generic Description of Telecommunication Services and Dynamic Resource Selection in Intelligent Communication Environments" by Stefan Arbanowksi ("Arbanowski96")	October 6, 1996	Ex. 11
Article entitled "Resource Selection in Heterogeneous Communication Environments using the Teleservice Descriptor" by Tom Pfeifer, Stefan Arbanowski, and Radu Popescu-Zeletin ("Pfeifer97")	December 19, 1997	Ex. 12
U.S. Patent No. 6,104,500 entitled "Networked Fax Routing Via Email" by Hassam Alam, Horace Dediu, and Scot Tupaj ("Alam")	April 29, 1998	Ex. 13
U.S. Patent No. 5,298,674 entitled "Apparatus for Discriminating an Audio Signal as an Ordinary Vocal Sound or Musical Sound" by Sang-Lak Yun ("Yun")	March 29, 1994	Ex. 14
U.S. Pat. No. 6,243,667 entitled "Network Flow Switching and Flow Data Export," by Darren R. Kerr and Barry L. Bruins ("Kerr")	May 28, 1996	Ex. 15
Article entitled "Cisco NetFlow Switching speeds traffic routing," InfoWorld Magazine ("NetFlow")	July 7, 1997	Ex. 16
Article entitled "A Concrete Security Treatment of Symmetric Encryption" by M. Bellare et al. ("Bellare97")	October 27, 1997	Ex. 17

Prior Art Reference	Prior Art Date	Exhibit
Article entitled "XOR MACs: New Methods for Message Authentication Using Finite Pseudorandom Functions" by Mihir Bellare, Roch Guerin, and Phillip Rogaway ("Bellare95")	1995	Ex. 18
Book entitled "Local Area Network Concepts and Products: Routers and Gateways" from IBM ("IBM96")	May 1996	Ex. 19
Article entitled "Checkpoint Firewall-1 White Paper, Version 2.0" ("Checkpoint")	September 1995	Ex. 20
U.S. Pat. No. 5,835,726 entitled "System for securing the flow of and selectively modifying packets in a computer network," by Shwed et al. ("Shwed")	December 15, 1993	Ex. 21
U.S. Pat. No. 6,651,099 entitled "Method and Apparatus for Monitoring Traffic in a Network" by Russell S. Dietz et al. ("Dietz")	June 30, 1999	Ex. 22
Article entitled "Dynamic Reconfiguration of Agent-Based Applications" by Luc Bellisard, Noel de Palma, and Michel Riveill ("Bellisard")	September 10, 1998	Ex. 23
Publication entitled "DTE Firewalls Phase Two Measurement and Evaluation Report" by Timothy L. Fraser et al. of Trusted Information Systems ("Fraser")	July 22, 1997	Ex. 24
Article entitled "Router Plugins: A Software Architecture for Next Generation Routers" by Dan Decasper et al. ("Decasper98")	September 4, 1998	Ex. 25
Specification entitled RFC 1825: "Security Architecture for the Internet Protocol" by R. Atkinson ("RFC 1825")	August 1995	Ex. 26
Specification entitled RFC 1829: "The ESP DES-CBC Transform" by P. Karn et al. ("RFC 1829")	August 1995	Ex. 27

Prior Art Reference	Prior Art Date	Exhibit
Specification entitled RFC 1883: "Internet Protocol, Version 6 (IPv6) Specification" by S. Deering and R. Hinden ("RFC 1883")	December 1995	Ex. 28
Book entitled "IPv6: The New Internet Protocol" by Christian Huitema ("Huitema")	October 28, 1997	Ex. 29
Article entitled "Crossbow: A Toolkit for Integrated Services over Cell Switched IPv6" by Dan Decasper et al. ("Decasper97")	May 29, 1997	Ex. 30
Dissertation entitled "Scout: A Path-Based Operating System" by David Mosberger ("Mosberger")	1997	Ex. 31
Article entitled "Implementing Communication Protocols in Java" by Bobby Krupczak <i>et. al</i> ("HotLava")	October 1998	Ex. 32
Article entitled "An Extensible Protocol Architecture for Application-Specific Networking" by Marc Fiuczynski <i>et. al</i> ("Plexus")	January 22, 1996	Ex. 33
Article entitled "ComScript: An Environment for the Implementation of Protocol Stacks and their Dynamic Reconfiguration" by Murhimanya Muhugusa <i>et. al</i>	December 1994	Ex. 34

Most of these prior art references were not cited or considered by the PTO during prosecution of the '163 patent and are not cumulative to the art of record in the original file. Only one of the references relied upon in this Request were cited during the prosecution of the '163 patent (*i.e.*, Mosberger). However, the finding of a "reasonable likelihood" under Section 312 is "not precluded by the fact that a patent or printed publication was previously cited by or to the Office or considered by the Office." 35 U.S.C. § 312(a).

A copy of each patent or printed publication relied upon in establishing each substantial new question of patentability is included with this Request as required by 37 C.F.R. § 1.915(b)(4). These references are cited in the accompanying Information Disclosure Statement and Form PTO/SB/08A.

Pfeifer96 was published by November 27, 1996, and is prior art under 35 U.S.C. § 102(a) and (b). See Ex. V07. Pfeifer96-C is another version of Pfeifer96 which is substantively identical to Pfeifer96 except for its figures being rendered in color. Pfeifer-96C bears the date November 25-27, 1996.

ISDN98 bears the date August 1998 and is prior art under 35 U.S.C. § 102(a) and (b).

Nelson was published on November 6, 1995. See Ex. V01 (document from United States Copyright Office Public Catalog showing date of publication). It is prior art under 35 U.S.C. § 102(a) and (b).

Cox was published on June 4, 1996. See Ex. V02 (document from United States Copyright Office Public Catalog showing date of publication). It is prior art under 35 U.S.C. § 102(a) and (b).

Franz98 bears the date April 22, 1998, and is prior art under 35 U.S.C. § 102(a) and (b).

Meer96 bears the date October 6, 1996, and is prior art under 35 U.S.C. § 102(a) and (b).

RFC 793 bears the date September 1981, and is prior art under 35 U.S.C. § 102(a) and (b).

Mills bears the date 1986, and is prior art under 35 U.S.C. § 102(a) and (b).

Arbanowski96 bears the date October 6, 1996, and is prior art under 35 U.S.C. § 102(a) and (b).

Alam was filed on April 22, 1998, and is prior art under 35 U.S.C. § 102(e).

Yun was issued on March 29, 1994 and is prior art under 35 U.S.C. § 102(a) and (b).

Kerr was filed on May 28, 1996 and is prior art under 35 U.S.C. § 102(e).

NetFlow bears the date July 7, 1997, and is prior art under 35 U.S.C. § 102(a) and (b).

Bellare97 bears the legend "IEEE 1997" and was published by October 22, 1997. See Ex. V03 (document from IEEE website showing date of publication). It is prior art under 35 U.S.C. § 102(a) and (b).

Bellare95 bears the date 1995, and is prior art under 35 U.S.C. § 102(a) and (b).

IBM96 bears the date May 1996, and is prior art under 35 U.S.C. § 102(a) and (b).

Checkpoint bears the date September 1995, and is prior art under 35 U.S.C. § 102(a) and (b).

Shwed was filed June 17, 1996 and issued November 19, 1998, and is prior art under 35 U.S.C. § 102(a) and (b).

Dietz was filed as a provisional application on June 30, 1999, and is prior art under 35 U.S.C. § 102(e).

Bellisard was published on September 10, 1998. See Ex. V04. It is prior art under 35 U.S.C. § 102(a) and (b).

Fraser bears the date July 22, 1997, and is prior art under 35 U.S.C. § 102(a) and (b).

Decasper98 bears the date September 1998, and was published by September 4, 1998. See Ex. V05. It is prior art under 35 U.S.C. § 102(a) and (b). Decasper98B&W bears the date 1998, and is a black & white version of Decasper98.

Decasper was published in 1998. See Ex. V06. It is prior art under 35 U.S.C. § 102(a) and (b).

RFC 1825 bears the date August 1995, and is prior art under 35 U.S.C. § 102(a) and (b).

RFC 1829 bears the date August 1995, and is prior art under 35 U.S.C. § 102(a) and (b).

RFC 1883 bears the date December 1995, and is prior art under 35 U.S.C. § 102(a) and (b).

Huitema was published on October 28, 1997. See Ex. V08 (document from United States Copyright Office Public Catalog showing date of publication). It is prior art under 35 U.S.C. § 102(a) and (b).

Decasper97 was published in 1997. See Ex. V09. It is prior art under 35 U.S.C. § 102(a) and (b).

Mosberger bears the date 1997, and is prior art under 35 U.S.C. § 102(a) and (b).

HotLava bears the date October 1998, and is prior art under 35 U.S.C. § 102(a) and (b).

Plexus bears the date January 22, 1996, and is prior art under 35 U.S.C. § 102(a) and (b).

ComScript was published in December 1994 and is prior art under 35 U.S.C. § 102(a) and (b). *See* Ex. 38 (document from publisher website indicating date of publication)

The following other written evidence is also made of record, solely to help explain the content of certain of the references listed in the table above. *See* MPEP § 2205.

Other Written Evidence	Exhibit
FH: Original Claims	Ex. 35-A
FH: 9/23/2002 Office Action	Ex. 35-B
FH: 2/24/2003 Amendment	Ex. 35-C
FH: 5/20/2003 Notice of Allowance	Ex. 35-D
FH: Ex Parte Reexamination Request	Ex. 35-E
FH: Order Granting Ex Parte Reexamination Request	Ex. 35-F
FH: 7/7/2009 Office Action	Ex. 35-G
FH: 9/1/2009 Amendment	Ex. 35-H
FH: 10/23/2009 Interview Summary	Ex. 35-I
FH: 12/4/2009 Final Office Action	Ex. 35-J
FH: 12/18/2009 Response to Final Rejection	Ex. 35-K
FH: 1/21/2010 Advisory Action	Ex. 35-L
FH: 2/8/2010 Amendment After Final	Ex. 35-M
FH: 3/2/2010 Notice of Intent to Issue Certificate	Ex. 35-N
First Amended Complaint	Ex. 36-A
Implicit Patent Infringement Contentions	Ex. 36-B
Implicit Infringement Claim Chart (Security Devices)	Ex. 36-C
Implicit Infringement Claim Chart (Application Acceleration)	Ex. 36-D
Implicit Opening Claim Construction Brief	Ex. 37-A

Other Written Evidence	Exhibit
Defendants Responsive Claim Construction Brief	Ex. 37-B
Implicit Reply Claim Construction Brief	Ex. 37-C
Implicit Technical Tutorial	Ex. 37-D
Defendants Technical Tutorial	Ex. 37-E
Technical Tutorial Transcript	Ex. 37-F
Implicit Claim Construction Slides	Ex. 37-G
Defendants Claim Construction Slides	Ex. 37-H
Claim Construction Transcript – Day 1	Ex. 37-I
Claim Construction Transcript – Day 2	Ex. 37-J
Penn State University, CiteSeer Digital Library	Ex. 38

IV. CLAIM CONSTRUCTION ADMISSIONS OF THE PATENT OWNER

A party requesting reexamination is permitted to submit admissions of the patentee in support of its request or proposed grounds for rejection. “The admission can reside in the patent file (made of record during the *prosecution* of the patent application) or may be presented during the pendency of the reexamination proceeding *or in litigation*.” MPEP 2617(III). Following is a brief description of the prosecution of the ‘163 patent (original and reexamination history), as well as statements by Implicit regarding claim construction in connection with its litigation against Requester.

Note that, both here and throughout this Request, the claims are accorded their broadest reasonable interpretation for purposes of reexamination only. Requester notes that claim construction in reexamination is broader than claim construction in litigation. *See In re*

Yamamoto, 740 F.2d 1569, 1571 (Fed. Cir. 1984). Therefore, nothing in this Request should be taken as an assertion regarding how the claims should be construed in litigation.³

A. Original Prosecution

During the original prosecution of the '163 patent, the patentee initially proposed 34 claims. Ex. 35-A [Original Claims] at 21-25. The PTO initially rejected all of these claims as being anticipated by at least three patents: U.S. Patent No. 5,870,479 to Feiken et al. ("Feiken"), U.S. Patent No. 5,425,029 to Hluchyj et al. ("Hluchyj"), and U.S. Patent No. 5,568,478 to Van Loo, Jr. et al. ("Van Loo"). Ex. 35-B [9/23/2002 Office Action] at 2-6. In response, the patentee cancelled those claims and proposed a new set of claims with additional language, including the "storing" step "so that the sequence *does not need to be re-identified* for *subsequent packets* of the message." Ex. 35-C [2/24/2003 Amendment] at 2. The patentee also offered a few arguments in an attempt to distinguish the cited prior art. *Id.* at 9-10. However, in issuing a notice of allowance for the new claims, the examiner appeared to rely primarily on the new limitations added to the claims. Ex. 35-D [5/20/2003 Notice of Allowance] at 2. The examiner further entered an examiner's amendment to the patent title, which was changed to: "Method and System for Demultiplexing a First Sequence of Packet Components to Identify Specific Components Wherein Subsequent Components are Processed Without Re-Identifying Components." *Id.*

³ Moreover, nothing in this Request should be construed as expressing any position as to whether the claims of the '163 patent claims constitute patentable subject matter under 35 U.S.C. § 101, or whether they satisfy the definiteness, enablement, best mode, or written description requirements of 35 U.S.C. § 112, since these grounds of invalidity cannot properly be raised in a request for reexamination. *See* MPEP § 2617 ("Other matters, such as . . . 35 U.S.C. 112 . . . will not be considered when making the determination on the request and should not be presented in the request."); *see also* MPEP § 2143.03(I) (even limitations rejected for indefiniteness must be examined).

B. Reexamination

On January 17, 2009, the PTO granted a request for *ex parte* reexamination of the '163 patent. Ex. 35-F [Order Granting Request for Ex Parte Reexamination]. Among other prior art references not considered during the original prosecution of the '163 patent, the PTO determined that a substantial new question of patentability existed based upon a 1997 doctoral dissertation by David Mosberger, entitled “Scout: A Path-Based Operating System” (“Mosberger”). The PTO subsequently issued an initial office action rejecting every single claim of the '163 patent as anticipated by Mosberger. Ex. 35-G [07/07/2009 Office Action] at 5-13.

Implicit initially attempted to distinguish Mosberger without making any substantive amendments to the claims. In its first office action response, Implicit argued that Mosberger “configures paths (formed from a sequence of components) before receiving the 'first packet of the message.” Ex. 35-H [09/01/2009 Amendment] at 11 (emphasis in original). In contrast, Implicit characterized the system claimed in the '163 patent as “configur[ing] paths at run-time (i.e., after the first packet is received).” *Id.* (emphasis in original). Implicit pointed to the first column of the '163 patent specification as “critical,” explaining that its claims required that sequence of components be “Created Dynamically”:

In other words, the '163 Patent clearly states that the invention requires the sequence of conversion routines (that form the paths) to be identified at run-time, and disavows prior art systems (like Mosberger) that use pre-configured paths, which are defined at “build-time” before the first packet of a message is received.

Id. at 18. Implicit also presented these and other arguments in an interview with the Examiner, along with a PowerPoint presentation. *See* Ex. 35-I [10/23/2009 Interview Summary].

The PTO initially rejected Implicit’s arguments, finding them to be “not persuasive.” In a final office action, the PTO argued (among other things) that the distinction upon which Implicit relied was not actually included in the claim language of the '163 patent. Ex. 35-J

[12/04/2009 Final Office Action] at 13-14 (claimed invention “not recited as being dynamic in nature”).

In response to the final office action, Implicit submitted an amendment that expressly added the “dynamically” language to the claims, as well as the phrase “after the first packet is received.” *See* Ex. 35-K [12/18/2009 Response to Final Rejection] at 10. Implicit claimed it was adding this language merely to “further clarify” the scope of the existing claims. *Id.* The PTO initially refused to enter these after-final amendments. *See* Ex. 35-L [1/21/2010 Advisory Action]. Another interview was conducted, and Implicit submitted additional proposed amendments a few days later, this time expressly inserting the “non-predefined” limitation. Ex. 35-M [2/8/2010 Amendment After Final].

After these additional amendments, the PTO finally removed its rejection of claims 1, 15, and 35 based on Mosberger. The PTO decision expressly relied on Implicit’s argument “that Mosberger does not dynamically identify sequences of components” Ex. 35-N [3/2/2010 Notice of Intent to Issue Ex Parte Reexamination Certificate] at 4.

C. Admissions Regarding Claim Construction

In addition to statements made during the prosecution history of the ‘163 patent, the patentee has made additional admissions regarding the scope and meaning of the claims in the allegations of its pleadings and infringement contentions prepared in connection with the concurrent litigation with Requester involving the ‘163 patent. *See* Ex. 36-A [Complaint]; Exs. 36-B – 36-D [Infringement Contentions]. Information regarding the patent owner’s apparent claim construction positions can be gleaned from these documents.

The patentee has also taken a number of express positions regarding claim construction in connection with *Markman* proceedings held in the concurrent litigation. The patentee presented

a technical tutorial describing the purported scope of the ‘163 patent claims, which is attached as Exhibit 37-D. The parties claim construction briefs are also attached as Exhibits 37-A – 37-C and are hereby incorporated by reference as if set forth herein.

For convenience, following is a chart summarizing the patent owner’s positions as set forth in its claim construction briefs:

Term	Implicit Proposed Construction
Non-predefined sequence of components	Sequence of components changeable at runtime
Dynamically identifying a sequence of components	Selecting at runtime a sequence of components
Input format	Structure or appearance of data to be processed
Output format	Structure or appearance of the data that results from processing
Selecting individual components	Selecting components that are not bound together by a compiler
Create/form [sequence of components]	Instantiate in memory
Processing [and all variants]	Manipulating data with a program
based on the first packet of the message	Plain meaning, no construction needed. In the alternative, relying on information in the first packet of the message
Identify ... a sequence of components ... such that the output format ... match[es] the input format of the next component	identify . . . a sequence of components . . . such that the output format is compatible with the input format of the next component
Message[s]	A collection or stream of data that is related in some way
State information	Information specific to a component for a specific message

The patentee has made additional express or implied admissions regarding claim meaning and scope regarding claim terms not presented to the Court in the concurrent litigation. For example, with respect to the term “demultiplexing,” the ‘163 patent states that “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.” Ex. 1 at 2:61-24; *see also* Ex. 36-A [Complaint] at 10 (demultiplexing performed “by reassembling datagrams fragmented over multiple packets”).

The patentee has made additional admissions of record in connection with the claim construction proceedings for the concurrent litigation. For example, at the claim construction hearing, Implicit provided the following statement regarding the meaning of “state information” in the context of the ‘163 patent:

As you process that message in '163, you look at the first packet, you figure out what it is, you figure out what it needs, you build your processing path, and then you keep track of the other packets that are related so you don't have to do that whole thing again.

The very essence of this system is to avoid the recursive packet-by-packet building a new data path every time. You build it once then you maintain state, which just means track what relates to that message so you can route the rest of the packets belonging to that message with the path that you have built.

That's what “maintaining state” means.

Ex. 37-J [1/19/12 Claim Construction Transcript] at 126.

These and other patentee admissions regarding claim construction (as set forth below) are applied in the analysis that follows as a reflection of what the patentee views as at least a reasonable construction of the claims at issue. Thus, for purposes of this Request, the “broadest reasonable construction” of the claims under consideration cannot be understood to be any narrower in scope than what for which the patentee itself has contended in litigation. Of course, application of the broadest reasonable construction in these proceedings should be not taken as an assertion or admission on the part of Requester regarding how the claims should be construed in litigation.

V. PERTINENCE AND MANNER OF APPLYING THE PRIOR ART

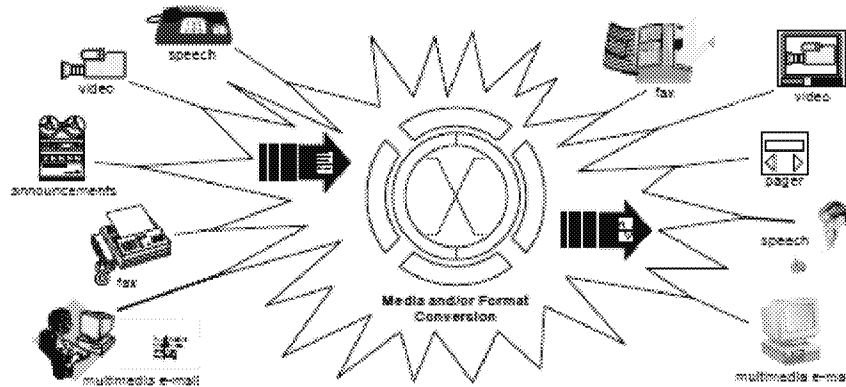
As shown in detail below, claims 1, 15, and 35 are invalid under 35 U.S.C. §§ 102 and 103 in light of the prior art references and combinations of references presented below. Requester respectfully submits that the analysis presented below satisfies the threshold requirement of showing a “reasonable likelihood that the requester would prevail with respect to at least 1 of the claims challenged in the request.” 35 U.S.C. § 312(a). The following proposed rejections should be adopted in their entirety.

For convenience in navigating the Request, following is a high-level summary of the base references on which rejections are based both alone and in combination with other references:

- Pfeifer96 and its obviousness combinations begin on page 24.
- Kerr and its obviousness combinations begin on page 112.
- Decasper98 and its obviousness combinations begin on page 177.
- Mosberger and its obviousness combinations begin on page 248.
- HotLava begins on page 269.

A. Pfeifer96 (Exhibit A02)

The article “Generic Conversion of Communication Media for Supporting Personal Mobility” by Tom Pfeifer and Radu Popescu-Zeletin (“Pfeifer96”) was published by November 27, 1996. Pfeifer96 is one of several references presented in this Request that describe what they term the “Intelligent Personal Communication Support System (iPCSS),” which is an “architecture” for supporting “personal mobility” to access “information any time, any place, in any form.” Ex. A02 at 105, 117.



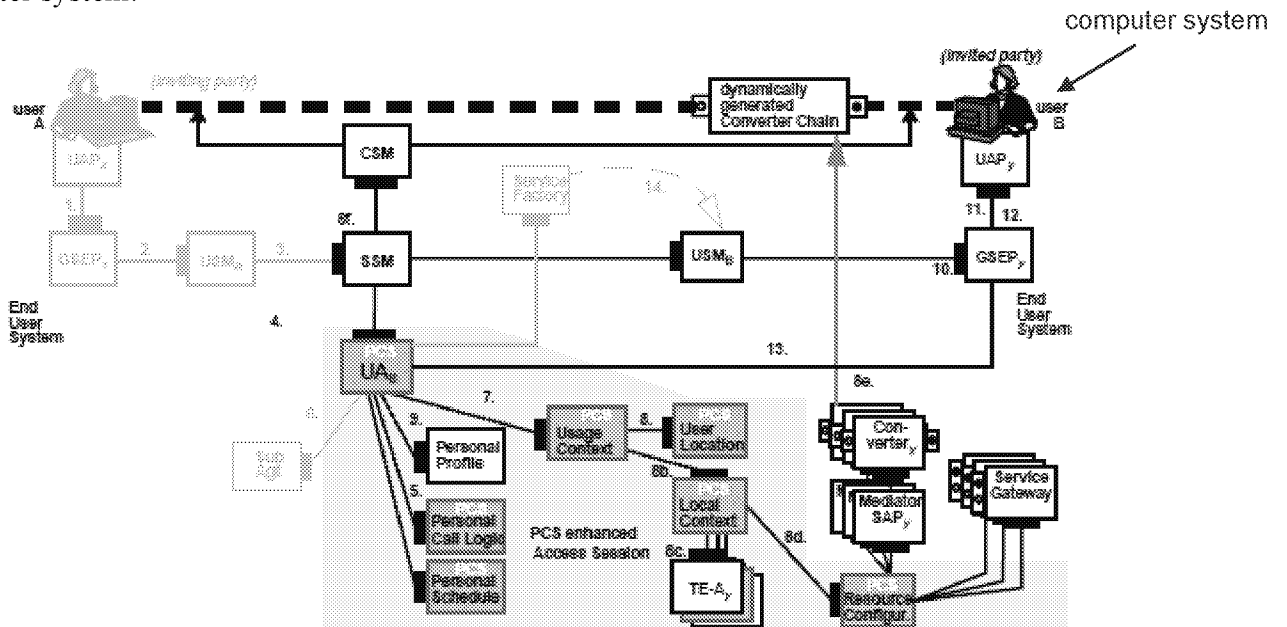
Id. at 118 (Figure 9: “Prioritized media conversion in the iPCSS”). None of these iPCSS references (including Pfeifer96) were considered during prosecution of the ‘163 patent.

1. Pfeifer96 Anticipates Claims 1, 15, and 35 Under § 102(a), (b)

(a) Claim 1

i. “A method . . . for processing a message”

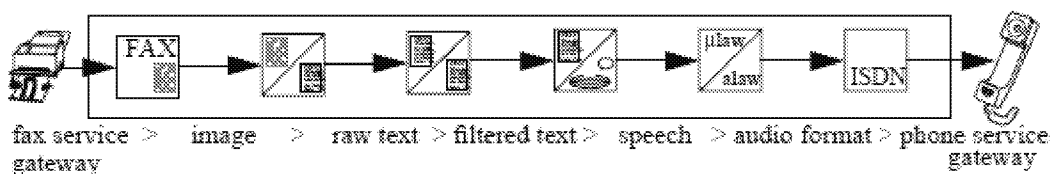
Claim 1 recites: “A method in a computer system for processing a message having a sequence of packets, the method comprising” Under Implicit’s apparent claim constructions, Pfeifer96 discloses this element. The IPCSS system is plainly illustrated as a computer system:



Id. at 122 (Figure 11: “Components of the PCS-enhanced TINA⁴ Access Session,” showing a “dynamically generated Converter Chain” connecting the two parties). Pfeifer96 teaches a “system/platform” implementing the “iPCSS architecture,” wherein communication between two parties over “fixed” and/or “wireless networks” is mediated by a “chain of converters” which is “dynamically generated.” Ex. A02 at 119, 105, 114, 124. Each “converter” in this chain may consist entirely of “software.” *Id.* at 113-14. Under Implicit’s apparent claim constructions, a “system/platform” capable of supporting such a software architecture would comprise “a computer system.”

Claim 1 further recites the method is “for processing a message having a sequence of packets.” Under Implicit’s apparent claim constructions, Pfeifer96 discloses this element.

Pfeifer96 discloses conversion of a message from one format to a different format, which constitutes “processing” as that term is used in the patent. Specifically, Pfeifer96 teaches “the controlled combination (concatenation) of various converters.” *Id.* at 105. For example, a chain of converters could convert an incoming fax transmission into audible speech delivered to an ISDN telephone:



Id. at 111 (Figure 6: “Converter chain: fax reception, conversion to text and speech, telephone delivery”).

⁴ The “Telecommunications Information Networking Architecture (TINA)” was “developed by a consortium which is currently the focus of worldwide attention.” *Id.* at 113. iPCSS is an enhancement to this architecture, adding “Personal Communications Support.” *Id.* at 105.

Pfeifer96 also discloses that a message may comprise a “series of packets.” For example, Pfeifer96 explicitly discloses connections using the well-known “ISDN” network standard, which one of ordinary skill in the art would understand is inherently packetized. *Id.* at 109, 111.⁵ A voice call over an ISDN network, as illustrated in Pfeifer96 (*see id.*) would thus comprise “a message having a sequence of packets.” In addition, Pfeifer96 discloses application in a “TCP/IP-based” environment, which one of ordinary skill in the art would again be aware is inherently packetized. *Id.* at 114 (“controller framework . . . controls . . . data flow via fire redirection, pipes or TCP/IP-based services”).⁶ Thus, for example, “reception and delivery of multimedia e-mail” in a TCP/IP-based environment would comprise “a message having a series of packets.” *Id.* at 118, 126. Other packet-based embodiments are also disclosed. *See, e.g., id.* at 126 (“sending and reception of . . . faxes”); *id.* at 105 (“mobility of the user in fixed networks and wireless networks”); *id.* at 118 (mobility “enabled by means of . . . wireless network interfaces and protocols (i.e. cordless, cellular and satellite) is fundamental for the provision of ubiquitous, global connectivity”).

In short, Pfeifer96 teaches a “universal platform” meant to achieve “universal connectivity” over both “fixed and wireless networks,” including communication of “message[s] having a sequence of packets.” *See id.* at 105, 120, 117-18.

ii. “a plurality of components”

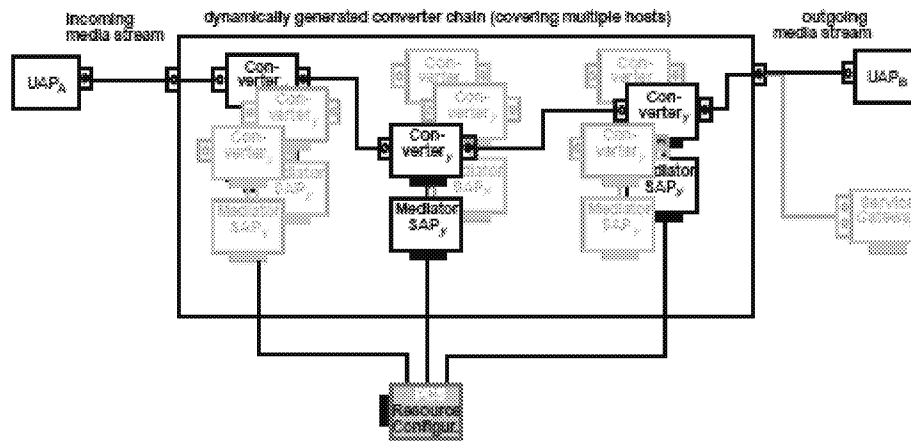
⁵ *See, e.g.,* Ex. X05 (“ISDN Primary Rate User-Network Interface Specification; Standard 08.01”) (August 1998) at 3-9 to 3-10 (Chapter 3-2: “Layer 2 frame structure,” where discrete frames with their own “Frame check sequence(s)” would comprise “packets” under Implicit’s apparent claim constructions. This reference is cited in this context solely to help explain Pfeifer96. *See* MPEP § 2205.

⁶ *See, e.g.,* Ex. X06 (RFC 791: “Internet Protocol [IP]” Specification) (1981) at 1 (“The Internet Protocol is designed for use in interconnected systems of packet-switched communication networks.”). This reference is cited in this context solely to help explain Pfeifer96. *See* MPEP § 2205.

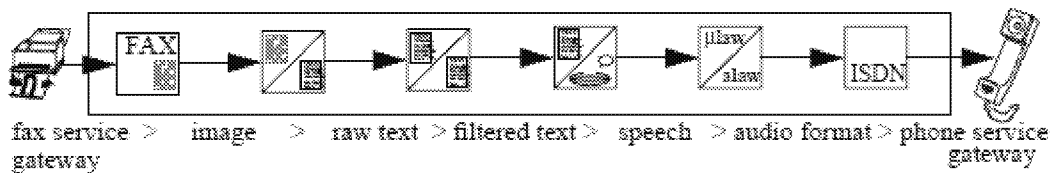
Claim 1 further recites: “providing a plurality of components, each component being a software routine for converting data with an input format into data with an output format.”

Under Implicit’s apparent claim constructions, Pfeifer96 discloses this element.

Pfeifer96 teaches “the controlled combination (concatenation) of various converters.” *Id.* at 105.



Id. at 125 (Figure 12: “Converter chain, configured for a specific task,” depicting a “dynamically generated converter chain” connecting an “incoming media stream” with an “outgoing media stream”). For example, a chain of converters could convert an incoming fax transmission into audible speech delivered to an ISDN telephone:



Id. at 111 (Figure 6: “Converter chain: fax reception, conversion to text and speech, telephone delivery”).

As apparent from the above example (where a fax is converted through a series of five intermediate formats to a telephone call), each converter converts “data with an input format into data with an output format.” This property of a converter is even presented in an abstract, formal

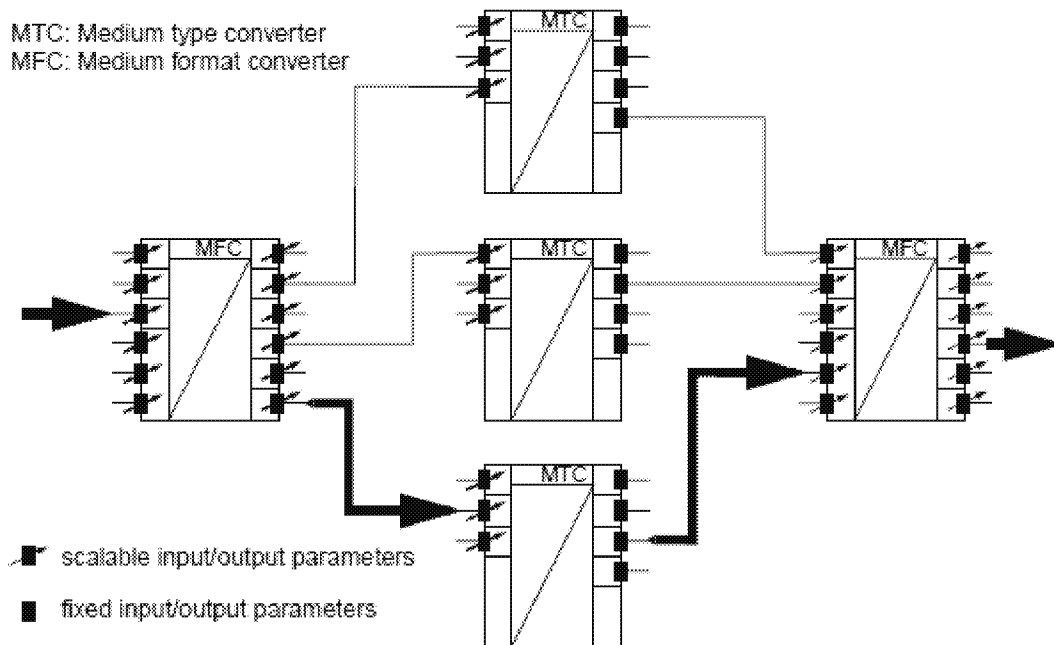
manner by Pfeifer96: “A media converter may be defined as a system entity, which input is information I_1 with the semantic S_1 , carried by a specific medium M_1 , using a specific form (or format) F_1 . We obtain information I_2 as output in another Medium M_2 in format F_2 , carrying a semantic S_2 (see Figure 1).” *Id.* at 3.



Id. at 105 (Figure 1: “Media converter system,” showing format alteration from F_1 to F_2).

More concretely, each converter performs either a more sweeping “Media type conversion” which alters the “medium type” (*e.g.*, “Text-to-Speech” or “Optical character recognition”), or a less sweeping “Media format conversion” which, *e.g.*, “converts into another format within the same type” (*e.g.*, video to video, but with a different “frame/sampling rate . . . resolution . . . [or] color depth”). *Id.* at 108, 125, 107. Under Implicit’s apparent claim construction, either type of conversion would “convert[] data with an input format into data with an output format,” because the data’s *format* is altered by the converter, and two mediums cannot share the same format: *e.g.*, text necessarily has a different format from audible speech, and audible speech necessarily has a different format from video.

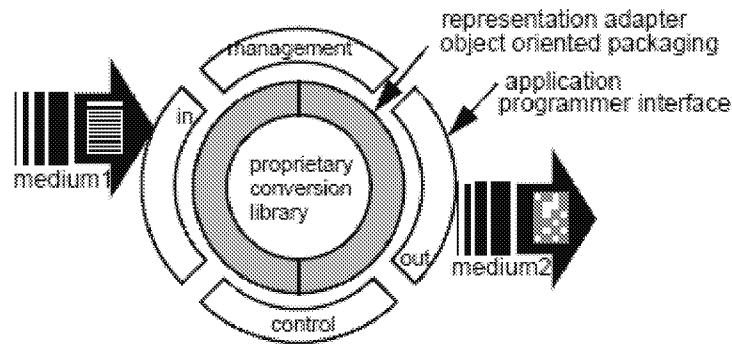
For example, the following diagram shows a chain of three converters chosen (“fat arrows”) for “a specific task of conversion.” *Id.* at 109. A first converter performs a medium format conversion (“MFC”), a second converter (chosen from a set of three possible converters) performs a medium type conversion (“MTC”), and a third converter performs another medium format conversion (MFC):



Id. at 108 (Figure 4: “Medium type conversion with format adaptation”). As discussed above, in all three cases (whether MTC or MFC), the data’s *format* is changed, because two mediums cannot share the same format.

This use of “chains of converters” which transform data through a series of intermediate formats is at the heart of the Pfeifer96 reference, and providing such chains clearly entails (as recited by claim 1) “providing a plurality of components” (each component comprising a converter) “for converting data with an input format into data with an output format.”

It is also clear that each component may comprise (as recited by claim 1) “a software routine.” Pfeifer96 teaches a “Generic Converter Model” wherein each converter component comprises at least two parts: (1) an “underlying” converter which may be comprised entirely of software such as a “proprietary conversion library”; and (2) a software wrapper *around* the underlying converter which provides a “Generic Converter Interface” for invoking it. *Id.* at 113-14, 109.



Id. at 113 (Figure 7: “Generic converter model,” showing an underlying converter (“proprietary conversion library” and software wrapper around it comprising a “representation adapter” and “application programmer interface”).

This “Generic Converter Interface” layer of software is needed around each underlying converter because the iPCSS employs a “Generic Converter Framework,” wherein converters from “different manufacturers” may be mixed and matched interchangeably to form “dynamically generated converter chain[s].” *See id.* at 113-14, 108, 122-23. And “[t]o form such a framework, the conversion tools utilized must” clearly “have unified interfaces.” *Id.* at 114.

Thus, each converter is a software routine with, moreover, a “unified interface[.]” for invoking it.

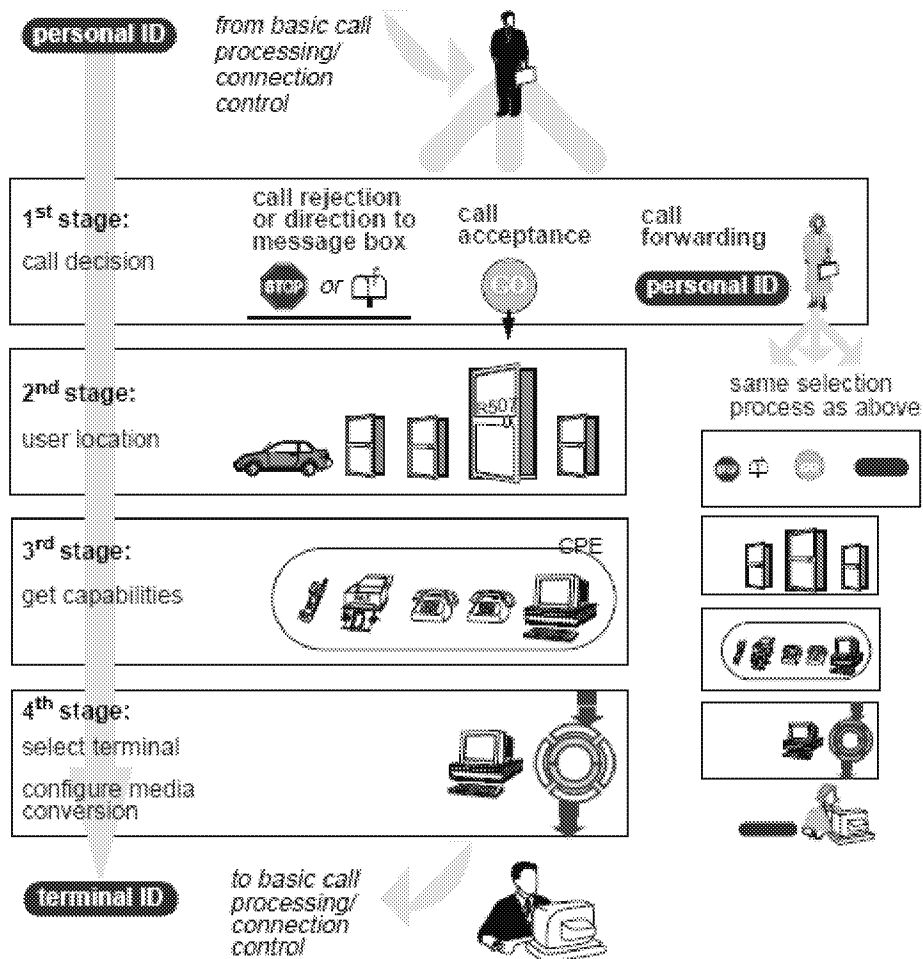
iii. “dynamically identifying a non-predefined sequence”

Claim 1 further recites: “for the first packet of the message, dynamically identifying a non-predefined sequence of components for processing the packets of the message such that the output format of the components of the non-predefined sequence match the input format of the next component in the non-predefined sequence, wherein dynamically identifying includes selecting individual components to create the non-predefined sequence of components after the

first packet is received.” Under Implicit’s apparent claim constructions, Pfeifer96 discloses this element.

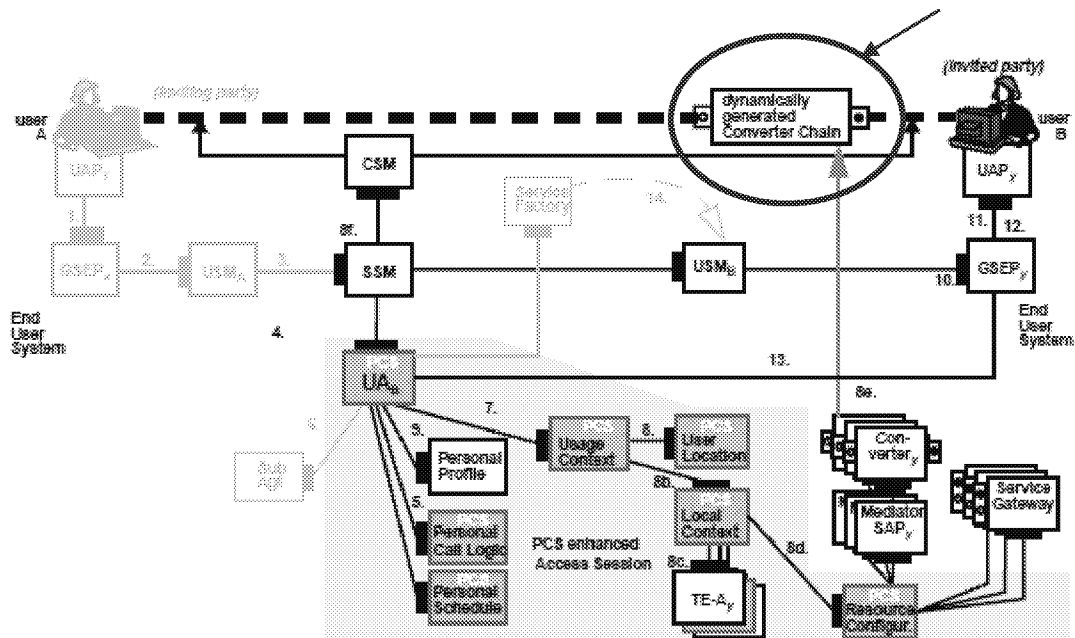
The iPCSS aims to “allow[] people to make use of any kind of terminal located at their whereabouts for obtaining access to their service.” *Id.* at 118. Because users are mobile and move in and out of range of various “terminal equipment” with varying “capabilities,” it is not possible to determine the specific media conversions that will be needed to achieve a connection to the user until the first packet of the message to that user has been received by the iPCSS. *See id.* at 119. This is why Pfeifer96 teaches a multi-stage call connection procedure, wherein the connection request⁷ is received in the “1st stage,” the called user’s “location” is ascertained in the “2nd stage,” the available pieces of terminal equipment at that location and their “capabilities” are ascertained in the “3rd stage,” and it is only in the “4th stage” that the specific “terminal” to accept the call will be selected, and an appropriate “converter chain” to achieve a connection to that terminal will be “**dynamically generated.**” *Id.* at 119, 124 (emphasis added).

⁷ One of ordinary skill would, of course, recognize that a connection request may comprise the first packet of a message. *See also* Section IV (“message”).



Id. at 119 (Figure 10: “PCS-based Intelligent Call Processing,” showing “select terminal” and “configure media conversion” in the “4th stage” of establishing a connection); *id.* at 120 (“4th . . . **Then**, the necessary converters are configured . . .”) (second emphasis added).

The following diagram provides a more granular (14-step) view of this call connection procedure as implemented by the iPCSS architecture:

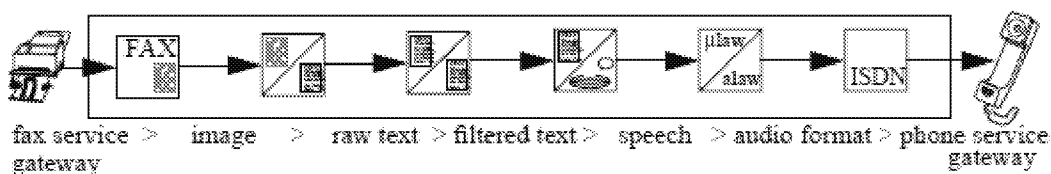


Id. at 122 (Figure 11: “Components of the PCS-enhanced TINA Access Session,” showing the connection request at step “1,” and the “**dynamically generated Converter Chain**” at end of step “8e”) (emphasis added).

Processing in the iPCSS proceeds in this order for an obvious reason: it is not possible to put together a suitable “chain of converters” between devices until the source and destination devices are *known*, and the devices will not be known before the first packet of the call has arrived. *See id.* at 114, 118-19. The iPCSS will not know the *source* device and its medium until the device initiates the call. *Id.* at 119-20. For example, is it a voice call, a fax, or an email? *Id.* at 119-20. And likewise the iPCSS will not know “the set” of possible *destination* devices in the called “user’s current vicinity” until the call is initiated, because the user’s vicinity (and hence the devices in that vicinity) can change from moment to moment. *Id.* For example, is there a fax machine or computer nearby, or merely a telephone? *See id.* As explained by Pfeifer96: “The iPCSS architecture . . . aim[s] . . . to increase the nomadic user’s reachability by introducing . . . the **dynamic** selection of terminals.” *Id.* at 122 (emphasis added).

Further confirming that, under Implicit’s apparent claim construction, Pfeifer96 teaches the *dynamic* generation of converter chains to terminals which are *dynamically* selected after initiation of a call, Pfeifer96 provides significant detail on the process by which an appropriate “converter chain” is generated at “runtime.” *Id.* at 124, 116.

Once a call has been placed, “the set of all access devices in the [called] user’s current vicinity” can be ascertained. *Id.* at 120. If none of these devices supports the “desired medium of the call” (*e.g.*, if there is no fax machine to accept a fax), then a component of the iPCSS called the “Resource Configurator” will systematically evaluate the various possible “chains of multiple converters” that could adapt the call to one of the available destination devices. *Id.* at 120, 124. Notably, the selection process does not merely involve picking one of a number of pre-assembled chains, but rather the system “*selects and configures one or multiple converters dynamically* to an appropriate converter chain.” *Id.* at 127 (emphasis added). At a high level, this can be seen as simply selecting and sequentially ordering a number of converters to create a chain of converters like the following to deliver a fax to a user having only a telephone in his current vicinity:



Id. at 111 (Figure 6: “Converter chain: fax reception, conversion to text and speech, telephone delivery”).

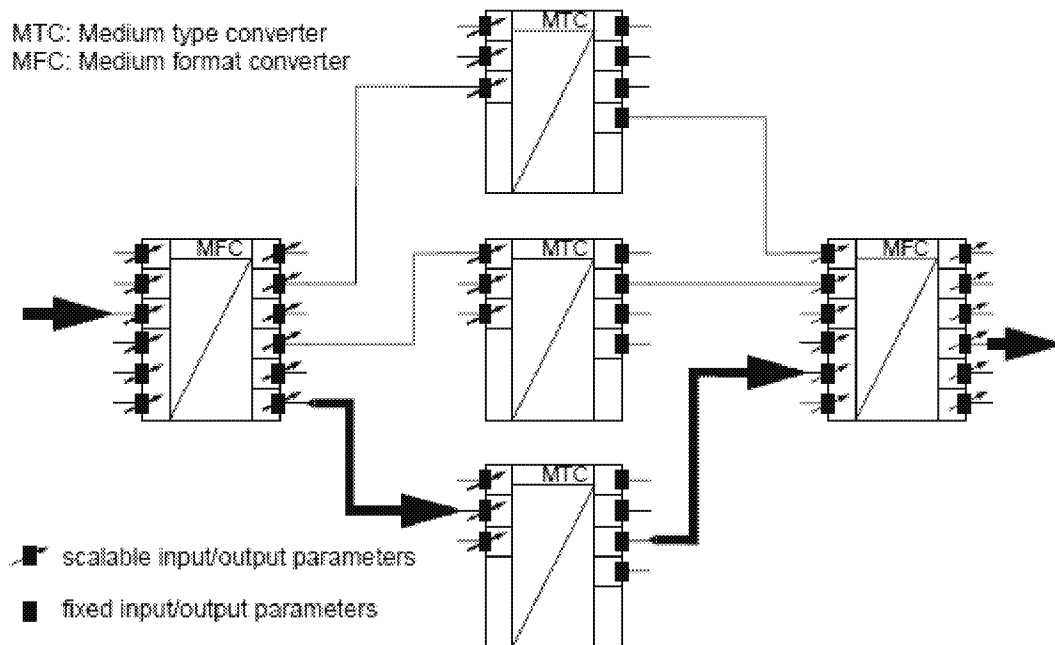
While finding a *workable* chain is certainly an aspect of the analysis performed by the Resource Configurator, Pfeifer96 explains that the analysis performed by this component goes even further. Specifically, the iPCSS performs a complex “Quality of Service” analysis which

takes into account the following factors (and others besides) when considering the various components that may be evaluated and ultimately assembled to form a chain of converters. *Id.* at 124, 114-16.

Pfeifer96 begins by recognizing that the various possible destination terminals in the called user's vicinity may have different "capabilities." *Id.* at 124. For example, one terminal may have only an "8 bit screen" while another has a higher resolution, and if "a 24 bit colour image" is "to be displayed," it would be preferable to use the latter. *See id.* at 115, 124.

Pfeifer96 also recognizes that the chain of converters necessary to reach a given terminal may result in some degradation of the message. *Id.* at 115. For example, there may be a loss in the "intelligibility" of a written fax message when it is read aloud by a Text-to-Speech converter, so such a chain requiring such a conversion may be less desirable if there are better alternatives. *See id.* at 115, 111.

Pfeifer96 further recognizes that each converter in a chain may have various settable input and output parameters (*e.g.*, controlling "frame/sampling rate, quantization, resolution, size, color depth . . . compression technique," and so on), and that the parameters chosen for the converters across any individual chain should be coordinated to avoid unnecessary loss of information. *See id.* at 107-08, 115, 124.



Id. at 108 (Figure 4: “Medium type conversion with format adaptation”), 6 (explaining the “fat arrows” in Figure 4 show a specific path of three converters using selected input/output parameters). For example, “multiple lossy compression and decompression processes” across a chain should be avoided “if possible,” with lossy compression performed “only finally, not as an intermediate step.” *Id.* at 115.

Considering all of the above factors and others, the Resource Configurator will: (1) “configure chains of multiple converters” to the various possible destination terminals; (2) “evaluate their Quality of Service”; and (3) create the “chain most appropriate” for completing the call. *See id.* at 124, 114-15. When it has finished this “complex evaluation” of the various possible combinations of converters, the “result is . . . a **dynamically generated converter chain** with stream interfaces to the appropriate media.” *Id.* at 124, 116 (emphasis added).

Pfeifer96 further states not only that the process of “[c]omparing different possibilities of concatenating converters for a specific task requires a complex evaluation of the quality parameters involved” but also that this “complex evaluation” must be “performed at *runtime*.”

Id. at 124, 116 (emphasis added). Of course, as demonstrated above, this elaborate analysis cannot even *begin* until the first packet of the message has been received by the iPCSS. Among other reasons, it is not until that moment that the system can know the desired source medium for the call, as well as the set of possible destination terminals in the called user's current vicinity which might possibly receive the call, which is information necessary in order to properly select the individual components in the proper sequential order to perform the required conversion. *See id.* at 119, 122, 124; *see also id.* at 128 (“creates an appropriate converter chain for the desired task”).

Once a sequence of “one or multiple converters” has been “select[ed] and configure[d]” to form an “appropriate converter chain,” that chain is “*instantiated* as an object” in memory for subsequent use by the system. *Id.* at 127.

Thus, Pfeifer96 teaches (as recited by claim 1) “for the first packet of the message, dynamically identifying a non-predefined sequence of components for processing the packets of the message.”

Pfeifer96 also teaches (as further recited by claim 1) that the sequence of components is arranged “such that the output format of the components of the non-predefined sequence match the input format of the next component in the non-predefined sequence.” *See* Claim 1(ii) above (showing “each component being a software routine for converting data with an input format into data with an output format”).

Pfeifer96 also teaches (as further recited by claim 1) that “dynamically identifying includes selecting individual components to create the non-predefined sequence of components after the first packet is received.” The very purpose of the “Generic Converter Framework” disclosed by Pfeifer96 is to permit *disparate* individual converters from possibly *different*

suppliers to be concatenated together on an “as needed” basis into chains. *E.g., id.* at 108 (“different manufacturers”), 114 (the various converters “must have unified interfaces”), 105 (“the controlled combination (concatenation) of various converters”). Pfeifer96 teaches the controlled combination of individual *converters*, not the controlled application of chains defined before the call even began.

After receiving the first packet of the message, Pfeifer96 concatenates individual converters to form *many* possible chains that might be used to connect the message’s two endpoints. *See id.* at 114, 124. The possible chains are compared to determine which would provide the best “Quality of Service,” and a single, optimal chain is finally chosen for the call. *See id.* at 115-16 (“Comparing different possibilities of concatenating converters for a specific task requires a complex evaluation of the quality parameters involved, **performed at runtime**”) (emphasis added), 124 (“The result . . . is a **dynamically generated** converter chain”) (emphasis added).

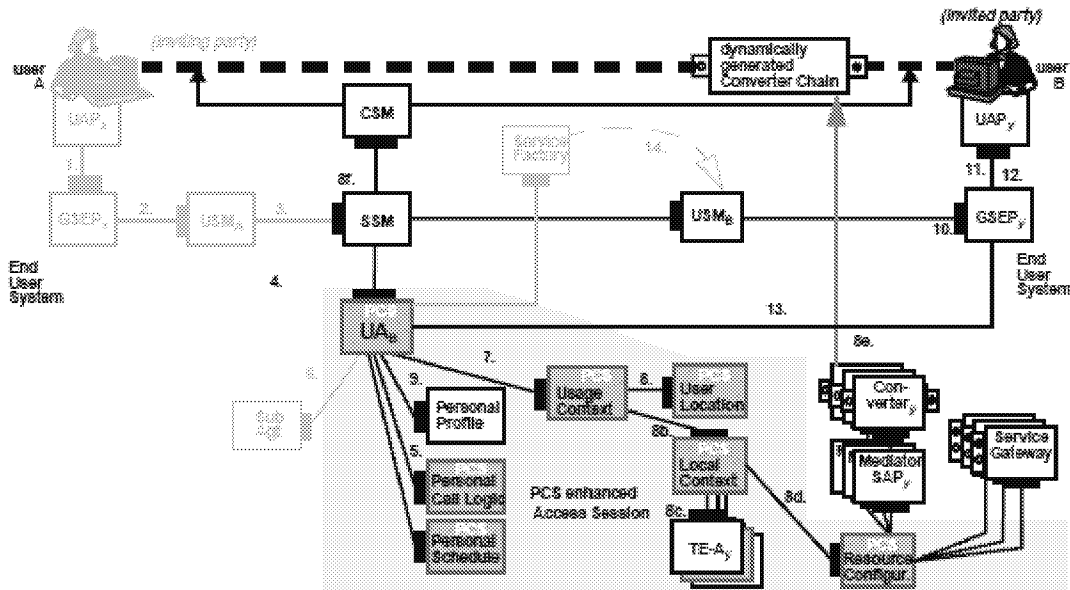
Because all of these possible chains are tailored to the specific message—*i.e.*, they connect *only* its two endpoints while using *only* mediums suitable for that message—none could be generated ahead of time. *See id.*

iv. “storing an indication of . . . the identified components”

Claim 1 further recites “storing an indication of each of the identified components so that the non-predefined sequence does not need to be re-identified for subsequent packets of the message.” Under Implicit’s apparent claim constructions, Pfeifer96 discloses this element.

Once the Resource Configurator has finished its “complex evaluation” to create a sequence of converters in a chain, that chain can be used in connection with a particular “session,” *i.e.*, a “grouping [of] specific activities in a service during a specific period of time.”

Id. at 120-21. The session or stream passes through the “dynamically generated converter chain.” *Id.* at 124, 116 (emphasis added).



Id. at 122 (Figure 11: “Components of the PCS-enhanced TINA Access Session,” showing the “dynamically generated Converter Chain” between the two parties). *See also* Section iv above.

This dynamic generation of a converter chain occurs just once for each “session,” and as implied by the existence of an incoming “stream interface[.]” to the chain, it is used to process the stream of packets from the source. *See id.* at 124, 122. *See also, e.g., id.* at 127 (the chain “is instantiated as an object with stream interfaces”) (emphasis added). In other words, it is “with respect to the possible converter chain” that “subsequent service processing establishes [a] stream connection” *Id.* at 127. Thus, every packet in a particular stream or session can make use of the “dynamically generated converter chain” without having to perform the “complex evaluation” for every single packet. *Id.* at 126-27.

v. “state information”

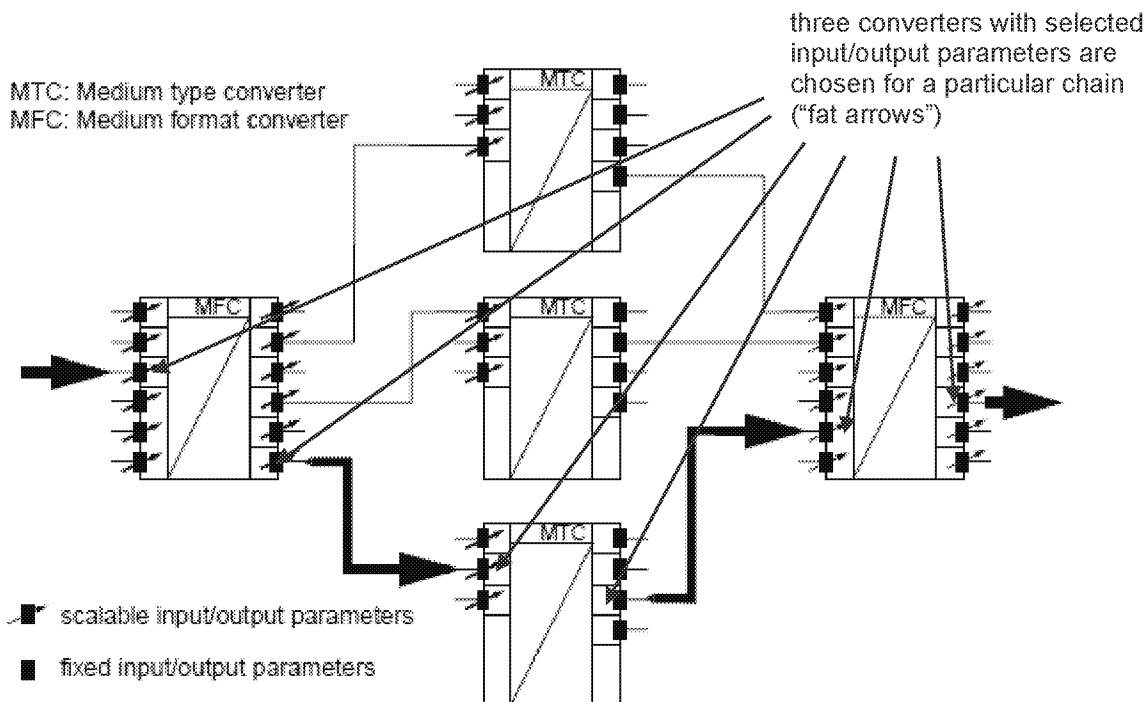
Claim 1 further recites “for each of a plurality of packets of the message in sequence, for each of a plurality of components in the identified non-predefined sequence, **retrieving** state

information relating to performing the processing of the component with the previous packet of the message; **performing** the processing of the identified component with the packet and the retrieved state information; and **storing** state information relating to the processing of the component with the packet for use when processing the next packet of the message.” Under Implicit’s apparent claim constructions, Pfeifer96 discloses this “state information” element.

As an initial matter, under Implicit’s apparent claim constructions, these “state information” elements would be satisfied merely by “storing an indication of each of the identified components so that the non-predefined sequence does not need to be re-identified” (as in Section vi above) and then using the stored indications to invoke the identified components in the sequence. *See* Section IV.C (“state information”). Pfeifer96 teaches such storing and such use, and thus it satisfies these “state information” elements as well. *See* Claim 1(iv) above.

However, Pfeifer96 also discloses these “state information” elements under a claim construction that requires component-by-component state information unrelated to the overall sequence of conversion routines.

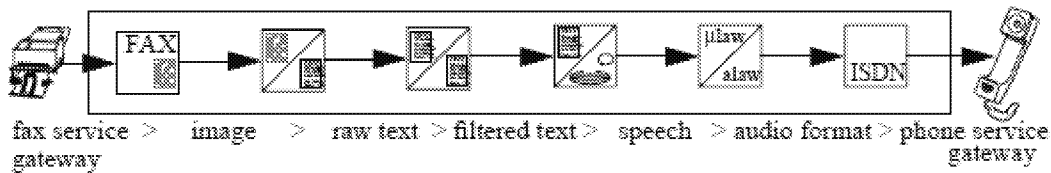
For example, Pfeifer96 teaches that as part of its “framework of type and format converters” (illustrated in Figure 5), each component would maintain multiple “scalable input/output parameters” which would comprise (as recited by claim 1) “state information relating to the processing of the component” under Implicit’s apparent claim constructions. Ex. A02 at 108-09.



Id. at 108 (Figure 4: “Medium type conversion⁸ with format adaptation”). As explained above, such parameters are considered when performing the Quality of Service analysis comparing many possible chains that might be used. *See* Section iv above. Pfeifer96 teaches these parameters control, *e.g.*, “frame/sampling rate, quantization, resolution, size, color depth . . . compression technique,” and so on), and that the parameters chosen for the converters across any specific chain should be coordinated to avoid unnecessary loss of information. *See id.* at 107-08, 115, 124; Section iv above. These parameters are “state information” which must be **stored** for each component, and which must be **retrieved** by each component in order to **perform** its processing: *e.g.*, in order to know which input or output “sampling rate” or “compression technique” to use when processing the packet. *See id.*

⁸ Medium *type* conversion (MTC) changes the format as well, since two different medium types cannot share the same format (*e.g.*, text vs. audible speech). *See* Section iii above.

As another and more specific example, Pfeifer96 teaches a chain of converters for delivering a fax to a user who has access to a telephone but not a fax machine:

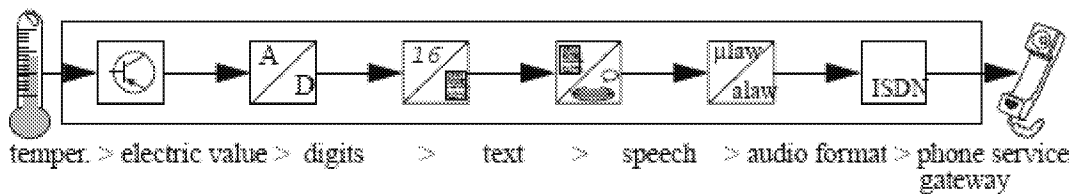


Id. at 111 (Figure 6: “Converter chain: fax reception, conversion to text and speech, telephone delivery”). It would be apparent to one of ordinary skill in the art that at least a plurality of the components in this chain would need to **store** and **retrieve** state information on a per packet basis in order to **perform** the processing described by Pfeifer96. For example, there is a component in the chain for adapting audio data to an ISDN phone connection. One of ordinary skill in the art would understand ISDN is a stateful protocol and that it would be necessary for this component to **store** and **retrieve** connection state on a per packet basis in order to correctly **perform** the processing for its side of the ISDN connection.⁹ As another example, there is a “Text-to-speech” (TTS) component in the chain, and Pfeifer96 teaches that such converters “analys[e] the grammatical structure of a sentence” in order to “improve the prosody of the speech, that is the intonation and phrase melody.” *Id.* at 111-12. Since the packet boundaries in an incoming stream of text will *not* generally be synchronized with the grammatical boundaries of sentences, this component would on a per packet basis need to **perform** the processing of monitoring for sentence boundaries and buffering accordingly (*e.g.*, **storing** a trailing incomplete

⁹ See, *e.g.*, Ex. X05 (“ISDN Primary Rate User-Network Interface Specification; Standard 08.01”) (August 1998) at 3-18 (“Transmitter send sequence number”; “Transmitter receive sequence number”; “The I format is used for frames that transfer information between Layer 3 entities”), 3-20 (“When using I-frame commands, each point-to-point data link connection endpoint has an associated send state variable” and “an associated acknowledge state variable”), 4-54 (“Call state”). This reference is cited in this context solely to help explain Pfeifer96. See MPEP § 2205.

sentence fragment until it can be **retrieved** to form complete sentence as subsequent packet(s) arrive). When a complete sentence is finally obtained, the component can then **perform** the additional processing of (1) assigning correct prosody to the sentence (*e.g.*, with sensitivity to its high and low points), and (2) creating the corresponding stream of audio for the utterance. Thus, it is clear that at least a plurality of components in this sequence (audio-to-ISDN and Text-to-Speech) would perform these “state information” operations, and would do so for each packet (not merely a plurality of packets).

As another example, Pfeifer96 teaches a chain of converters for “reading temperature values over a phone line” to a user with an ISDN telephone. Ex. A02 at 109.



Id. (Figure 5: “Converter chain for temperature to speech conversion with telephone delivery”).

As with the fax example above, this sequence contains both an audio-to-ISDN component and a Text-to-Speech component, and thus it is clear that for this example as well, at least a plurality of components in the sequence would perform the “state information” operations, and would do so for each packet (not merely a plurality of packets).

Thus, it is clear from several perspectives and multiple examples that Pfeifer96 discloses this “state information” claim element.

(b) *Claim 15*

i. *“demultiplexing packets of messages”*

Claim 15 recites: “A method in a computer system demultiplexing packets of messages.”

Under Implicit’s apparent claim constructions, Pfeifer96 discloses this element. *See* Claim 1(i)

above (showing “A method in a computer system for processing a message having a sequence of packets”). Under Implicit’s apparent claim constructions, “demultiplexing” a packet is satisfied by routing a packet to the correct sequence of components for processing it—and it was obvious for Pfeifer96 to perform this function. *See* Section IV (“demultiplexing”) above and Claim 1(iii) above (showing “dynamically identifying a non-predefined sequence of components for processing the packets of the message”).

ii. “dynamically identifying a non-predefined sequence”

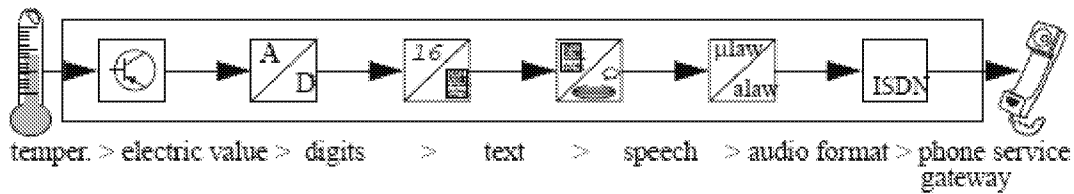
Claim 15 further recites “dynamically identifying a non-predefined sequence of components for processing each message based on the first packet of the message so that subsequent packets of the message can be processed without re-identifying the components, wherein dynamically identifying includes selecting individual components to create the non-predefined sequence of components after the first packet is received.” Under Implicit’s apparent claim constructions, Pfeifer96 discloses this element. *See* Claim 1(iii) (showing “for the first packet of the message, dynamically identifying a non-predefined sequence of components for processing the packets of the message”) and Claim 1(iv) (showing “storing an indication of each of the identified components so that the non-predefined sequence does not need to be re-identified for subsequent packets of the message”) above.

iii. “different . . . sequences of components can be identified”

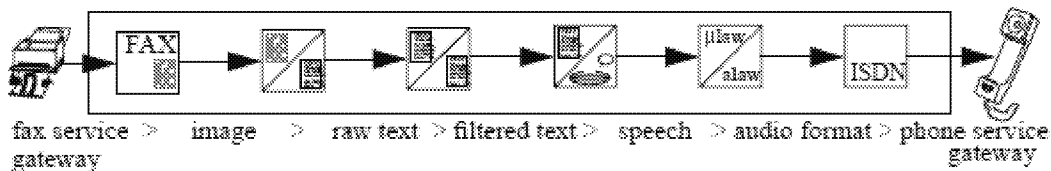
Claim 15 further recites “different non-predefined sequences of components can be identified for different messages.” Under Implicit’s apparent claim constructions, Pfeifer96 discloses this element.

Since both the source and destination and the source and destination mediums will vary across messages, it is clear that correspondingly different sequences of converters will generally

be required. *See, e.g.*, Ex. A02 at 109, 111. For example, a first communication may extend from a temperature sensor to an ISDN telephone:



Id. at 109 (Figure 5: “Converter chain for temperature to speech conversion with telephone delivery”). And a second communication may extend from a fax machine to an ISDN telephone, and it will therefore have a correspondingly different sequence of converters:



Id. at 111 (Figure 6: “Converter chain: fax reception, conversion to text and speech, telephone delivery”).

iv. “each component being a software routine”

Claim 15 further recites “each component being a software routine.” Under Implicit’s apparent claim constructions, Pfeifer96 discloses this element. *See* Claim 1(ii) (showing same element) above.

v. “selecting individual components”

Claim 15 further recites “dynamically identifying includes selecting individual components to create the non-predefined sequence of components.” Under Implicit’s apparent claim constructions, Pfeifer96 discloses this element. *See* Claim 1(iii) (showing same element) above.

vi. “state information”

Claim 15 finally recites “for each packet of each message, performing the processing of the identified non-predefined sequence of components of the message wherein state information generated by performing the processing of a component for a packet is available to the component when the component processes the next packet of the message.” Under Implicit’s apparent claim constructions, Pfeifer96 discloses this element. *See* Claim 1(v) (showing similar element) above.

(c) *Claim 35*

i. *“instructions for demultiplexing packets of messages”*

Claim 35 recites: “A computer-readable medium containing instructions for demultiplexing packets of messages.” Under Implicit’s apparent claim constructions, Pfeifer96 discloses this element. One of ordinary skill would understand that the software of Pfeifer96 (including its sequences of software-based converters) would be stored on a “computer-readable medium.” *See* Claim 1(i) above (showing “a computer system”).

ii. *“dynamically identifying a . . . non-predefined sequence”*

Claim 35 recites: “dynamically identifying a message-specific non-predefined sequence of components for processing the packets of each message upon receiving the first packet of the message.” Under Implicit’s apparent claim constructions, Pfeifer96 discloses this element. *See* Claim 1(iii) (showing “for the first packet of the message, dynamically identifying a non-predefined sequence of components for processing the packets of the message”) above.

iii. *“subsequent packets . . . can use the . . . sequence”*

Claim 35 further recites that “subsequent packets of the message can use the message-specific non-predefined sequence identified when the first packet was received.” Under Implicit’s apparent claim constructions, Pfeifer96 discloses this element. *See* Claim 1(iv)

(showing “storing an indication of each of the identified components so that the non-predefined sequence does not need to be re-identified for subsequent packets of the message”) above.

iv. “selecting individual components”

Claim 35 further recites that “dynamically identifying includes selecting individual components to create the message-specific non-predefined sequence of components.” Under Implicit’s apparent claim constructions, Pfeifer96 discloses this element. *See* Claim 1(iii) (showing “dynamically identifying includes selecting individual components to create the non-predefined sequence of components after the first packet is received”) above.

v. “state information”

Claim 35 further recites “for each packet of the message, invoking the identified non-predefined sequence of components in sequence to perform the processing of each component for the packet wherein each component saves message-specific state information so that that component can use the save message-specific state information when that component performs its processing on the next packet of the message.” Under Implicit’s apparent claim constructions, Pfeifer96 discloses this element. *See* Claim 1(v) (showing similar element) above.

2. Pfeifer96 Renders Obvious Claims 1, 15, and 35 Under § 103

If certain aspects recited in claims 1, 15, and 35 of the ‘163 patent are not deemed to be disclosed or inherent over Pfeifer96, then the inclusion of those aspects certainly would be obvious over Pfeifer96, under 35 U.S.C. § 103.

(a) Claim 1

i. “A method in a computer system . . .”

Claim 1 recites: “A method in a computer system for processing a message having a sequence of packets, the method comprising” Under Implicit’s apparent claim constructions, Pfeifer96 renders obvious this element.

Pfeifer96 teaches a “system/platform” implementing the “iPCSS architecture,” wherein communication between two parties over “fixed” and/or “wireless networks” is mediated by a “chain of converters” which is “dynamically generated.” Ex. A02 at 119, 105, 114, 122, 124. It was obvious that a “computer system” would be used to perform the various mediation functions disclosed by Pfeifer96, including, *e.g.*, the “dynamically generated Converter Chain”. *Id.* at 122, 113-14.

Claim 1 further recites the method is “for processing a message having a sequence of packets.”

As an initial general matter informing all of the argument below, Pfeifer96 teaches a “universal platform” meant to achieve “universal connectivity” over both “fixed and wireless networks.” *Id.* at 105, 120, 117-18. Thus, it was obvious to support incoming communications from at least any mainstream device over at least any mainstream communications medium, including over the main communications mediums which are inherently packetized.

More specifically, Pfeifer96 discloses “Service Gateways” which are “tools responsible for transporting information into and out of the context of the iPCSS, i.e., connecting the iPCSS to the world outside the TINA platform.” *Id.* at 126. These gateways must “consider the specific properties of the connected information and communication services.” *Id.*

For example, Pfeifer96 explicitly discloses a Service Gateway for “voice connection with the public telephone network,” for both incoming (“*phone.in*”) and outgoing (“*phone.out*”) voice connections. *Id.* at 126 (emphasis in original). Because Pfeifer96 repeatedly discloses voice connections using the well-known “ISDN” public telephone network standard, it was obvious that incoming voice connections might be placed from ISDN telephones and therefore that the Service Gateway should be capable of handling such incoming calls. *Id.* at 109, 111. One of

ordinary skill in the art would be aware that ISDN is inherently packetized,¹⁰ and that therefore an incoming voice call over an ISDN public telephone network would comprise “a message having a sequence of packets.”

Relatedly, Pfeifer96 also discloses a Service Gateway for “sending and reception of . . . faxes.” *Id.* at 126. One of ordinary skill in the art would be aware that fax machines are also commonly positioned on ISDN networks and that certainly ISDN networks are *capable* of transmitting fax data. Therefore it was obvious that the Service Gateway should be capable of handing such incoming and outgoing fax communications. Because ISDN is inherently packetized, each of these transmissions would comprise “a message having a sequence of packets.”

Relatedly, iPCSS discloses communication over wireless networks. *See id.* at 105 (“mobility of the user in fixed networks and wireless networks”), 118 (mobility “enabled by means of . . . wireless network interfaces and protocols (i.e. cordless, cellular and satellite) is *fundamental for the provision of ubiquitous, global connectivity*”) (emphasis added). Because many devices are accessible only via wireless networks, it was obvious that the system should support communications to and from such devices. One of ordinary skill would recognize wireless communications are inherently packetized, and that therefore any incoming communication over a wireless network would comprise “a message having a series of packets.”

As another example, Pfeifer96 discloses a Service Gateway for “reception and delivery of multimedia e-mail.” *Id.* at 126. Since email is at least predominantly transmitted over packet-

¹⁰ *See, e.g.*, Ex. X05 (“ISDN Primary Rate User-Network Interface Specification; Standard 08.01”) (August 1998) at 3-9 to 3-10 (Chapter 3-2: “Layer 2 frame structure,” where discrete frames with their own “Frame check sequence(s)” would comprise “packets” under Implicit’s apparent claim constructions. This reference is cited in this context solely to help explain Pfeifer96. *See* MPEP § 2205.

oriented networks (*e.g.*, using the TCP/IP protocol suite), it was obvious that the system should support such incoming communications, and any such incoming communication would comprise “a message having a series of packets.” *See also id.* at 118 (Figure 9, showing “multimedia e-mail” transmitted by a user at a computer).

As another example, Pfeifer96 discloses a Service Gateway “for support of multimedia conferencing.” *Id.* at 126. One of ordinary skill in the art would recognize that while it is possible to transmit *one* medium over a non-packetized, analog phone line (*i.e.*, audio), at least the *predominant* method of transmitting multimedia conferencing information is over a packet-oriented network (such as one using TCP/IP). It was obvious that the system should support such incoming and outgoing multimedia conferencing communications, and any such incoming communication would comprise “a message having a series of packets.” *See also id.* at 118 (Figure 9, showing “multimedia e-mail” transmitted by a user at a computer).

More generally, Pfeifer96 discloses various devices that the system would communicate with. *See, e.g., id.* at 110 (“Braille output device”), 6 (“temperature” sensor), 118 (“video” camera). It was obvious that because such devices typically do not have an independent communications capability of their own, these devices are typically connected to computers which can communicate with the outside world by use a packet-oriented network (such as one using TCP/IP). Thus, it was obvious that the system should support communications to and from these devices over packet-oriented networks, and any incoming communication from such a device would comprise “a message having a series of packets.”

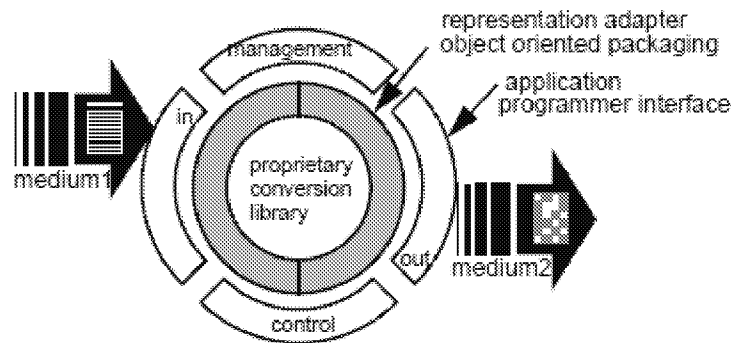
As a general matter, all of these specific instances of connectivity follow from the teaching of a “universal platform” meant to achieve “universal connectivity” over both “fixed and wireless networks.” *Id.* at 105, 120, 117-18.

ii. “a plurality of components”

Claim 1 further recites: “providing a plurality of components, each component being a software routine for converting data with an input format into data with an output format.”

Under Implicit’s apparent claim constructions, Pfeifer96 renders obvious this element.

As explained above, each converter component converts data with an input format into data with an output format, and each is invoked via an “application programmer interface”:



Id. at 113 (Figure 7: “Generic converter model”). While iPCSS-01 discusses “object oriented packaging” as an element of these converter components, it is at least obvious that object oriented software would comprise a “software routine,” or alternatively at least obvious to organize object oriented software such that it would comprise a software routine.

iii. “dynamically identifying a non-predefined sequence”

Claim 1 further recites: “for the first packet of the message, dynamically identifying a non-predefined sequence of components for processing the packets of the message such that the output format of the components of the non-predefined sequence match the input format of the next component in the non-predefined sequence, wherein dynamically identifying includes selecting individual components to create the non-predefined sequence of components after the first packet is received.” Under Implicit’s apparent claim constructions, Pfeifer96 renders obvious this element.

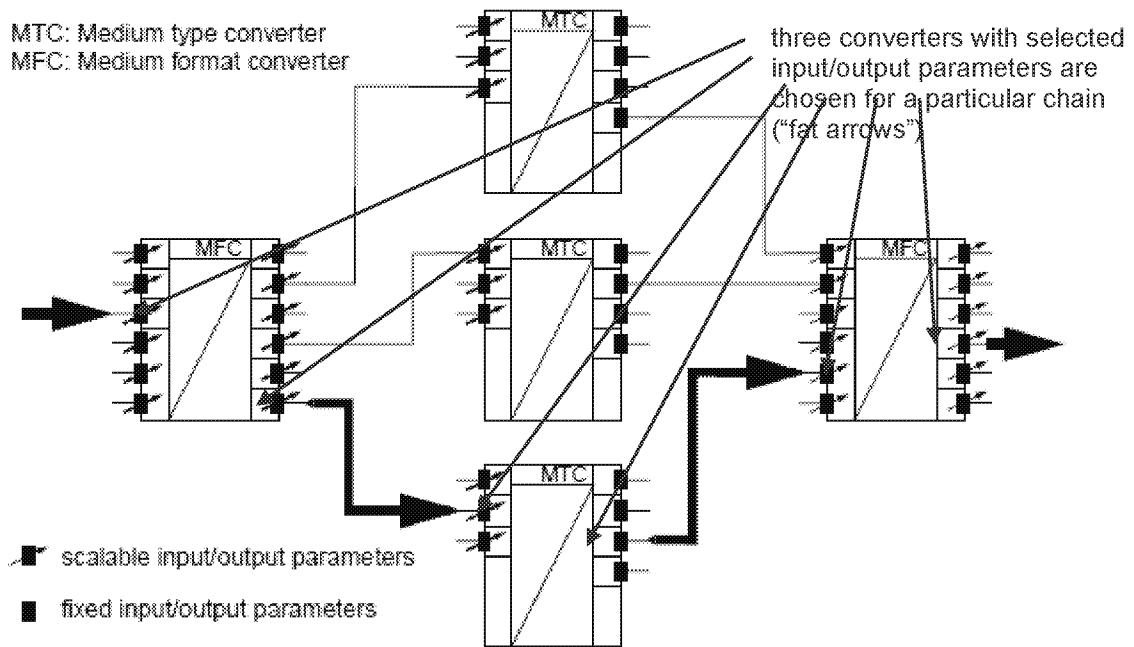
Claim 1 further recites “storing an indication of each of the identified components so that the non-predefined sequence does not need to be re-identified for subsequent packets of the message.” Under Implicit’s apparent claim constructions, Pfeifer96 renders obvious this element.

The “dynamically generated converter chain” is generated only once for an incoming communication and is used to process the entire communication. *See id.* at 122-27. It is therefore clear this chain is stored, and also obvious that it would be stored. For example, “instantiat[ing]” the entire chain “as an object with stream interfaces” (for accepting the incoming stream of packets) would comprise storing an indication of the components within it. *Id.* 127.

v. “state information”

Claim 1 further recites “for each of a plurality of packets of the message in sequence, for each of a plurality of components in the identified non-predefined sequence, **retrieving** state information relating to performing the processing of the component with the previous packet of the message; **performing** the processing of the identified component with the packet and the retrieved state information; and **storing** state information relating to the processing of the component with the packet for use when processing the next packet of the message.” Under Implicit’s apparent claim constructions, Pfeifer96 renders obvious this “state information” element.

As an initial matter, Pfeifer96 teaches that as part of its “framework of type and format converters” (illustrated in Figure 5), each component would maintain multiple “scalable input/output parameters” which would comprise (as recited by claim 1) “state information relating to the processing of the component” under Implicit’s apparent claim constructions. Ex. A02 at 108-09.



Id. at 108 (Figure 4: “Medium type conversion¹¹ with format adaptation”). As explained above, such parameters are considered when performing the Quality of Service analysis comparing many possible chains that might be used. *See* Claim 1(iii) above. Pfeifer96 teaches these parameters control, *e.g.*, “frame/sampling rate, quantization, resolution, size, color depth . . . compression technique,” and so on), and that the parameters chosen for the converters across any specific chain should be coordinated to avoid unnecessary loss of information. *See id.* at 107-08, 115, 124; Claim 1(iii) above. These parameters are “state information” which must be **stored** for each component, and which must be **retrieved** by each component in order to **perform** its processing: *e.g.*, in order to know which input or output “sampling rate” or “compression technique” to use when processing the packet. *See id.* It was obvious that any or all of the components in any particular chain would maintain such “parameters,” and therefore obvious that any or all of the components would satisfy the “state information” element of claim 1.

¹¹ Medium *type* conversion (MTC) changes the format as well, since two different medium types cannot share the same format (*e.g.*, text vs. audible speech). *See* Section iii above.

Additionally and more specifically, the most obvious implementations of a number of the components disclosed by Pfeifer96 would read on this claim element, and it was obvious that such components would comprise at least “a plurality of the components” in various obvious (and expressly disclosed) sequences.

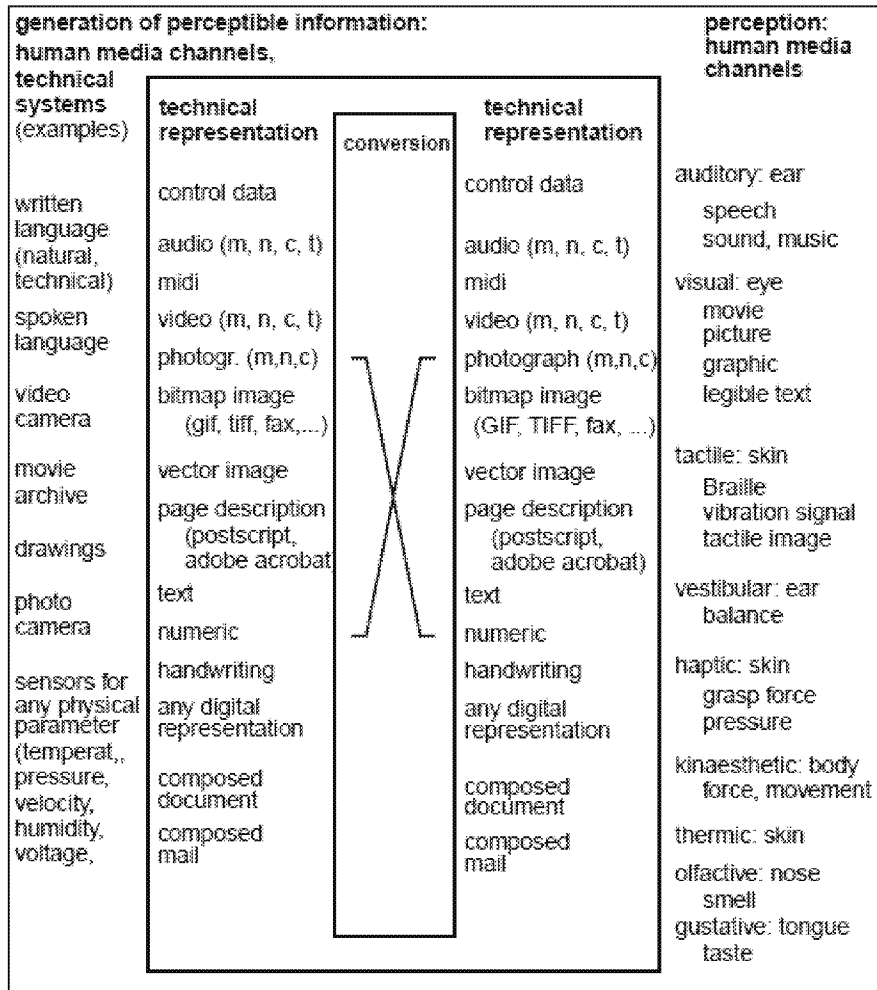
(a) ISDN Adapter Components

Pfeifer96 teaches a component for adapting between audio data and an ISDN telephone connection. Ex. A02 at 109, 111. As explained above, one of ordinary skill in the art would understand ISDN is a stateful protocol, and hence it would be obvious for this component to **store** and **retrieve** connection state on a per packet basis in order to correctly **perform** the processing for its side of the ISDN connection.¹² It was further obvious that such an ISDN adapter would be used for adapting either incoming and/or outgoing ISDN calls.

(b) Lossless Compression/Decompression Components

Pfeifer96 teaches that many of its converters would have a settable parameter “c” specifying the “applied compression [and decompression] technique” employed by the component as part of its conversion:

¹² See, e.g., Ex. X05 (“ISDN Primary Rate User-Network Interface Specification; Standard 08.01”) (August 1998) at 3-18 (“Transmitter send sequence number”; “Transmitter receive sequence number”; “The I format is used for frames that transfer information between Layer 3 entities”), 3-20 (“When using I-frame commands, each point-to-point data link connection endpoint has an associated send state variable” and “an associated acknowledge state variable”), 4-54 (“Call state”). This reference is cited in this context solely to help explain Pfeifer96. See MPEP § 2205.



parameters:
m, n: media dependent parameters
(frame/sampling rate, quantization, resolution, size, color depth, etc.)
c: applied compression technique
t: time, duration, etc.

Id. at 107 (Figure 3: “Generic conversion matrix”). Since Pfeifer does not cite specific compression/decompression techniques (beyond observing some are “lossy”), one of ordinary skill would draw from standard background knowledge regarding the range of compression/decompression techniques that might be applied.

Because Pfeifer96 cautions that “multiple *lossy* compression and decompression” across the same converter chain should be “prohibit[ed] . . . if possible,” one of ordinary skill would be motivated to apply lossless compression/decompression techniques when implementing these converters. *Id.* at 115, 124 (emphasis added). In particular, one of ordinary skill would be aware

that “adaptive” algorithms are among the leading lossless compression/decompression techniques available, and obvious implementations of such “adaptive” algorithms would entail maintaining “state information” across packets in the manner specified by claim 1. This standard background knowledge is confirmed by citation to Ex. 5 (“The Data Compression Book” by Mark Nelson et al., “Nelson”) (1996), which is cited in this context solely to help explain Pfeifer96. *See* MPEP § 2205.

Nelson explains “Adaptive coding . . . lead[s] to vastly improved compression ratios,” and that “compression research in the last 10 years has concentrated on adaptive models.” Ex. 5 at 8, 18. Adaptive algorithms include such well-known algorithms as “Adaptive Huffman Coding” (chapter 4; *id.* at 75), “Adaptive [Statistical] Modeling” (chapter 6; *id.* at 155), “[Adaptive] Dictionary-Based Compression” (chapter 7; *id.* at 203), and “Sliding Window Compression” (chapter 8; *id.* at 215); and the prominent “LZ” family of compression algorithms (chapter 8 and 9, *id.* at 221, 255). All of these adaptive techniques are lossless. *See id.* at 9 (“All of the compression techniques discussed through chapter 9 are ‘lossless’”).

Nelson further explains the stateful manner in which adaptive coding operates: “When using an adaptive model, data does not have to be scanned once before coding in order to generate statistics [used to perform compression/decompression]. Instead, the statistics are **continually modified as new characters are read in and coded**. The general flow of a program using an adaptive model looks something like that shown in Figures 2.2 and 2.3.” *Id.* at 18 (emphasis added).

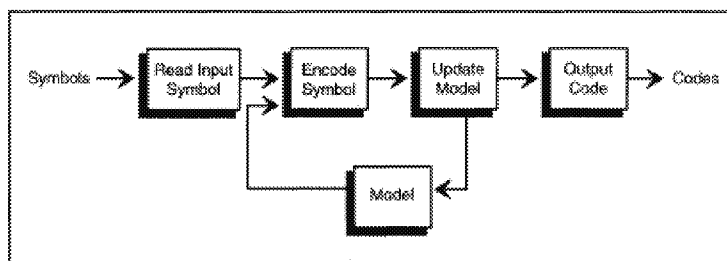


FIGURE 2.2 GENERAL ADAPTIVE COMPRESSION.

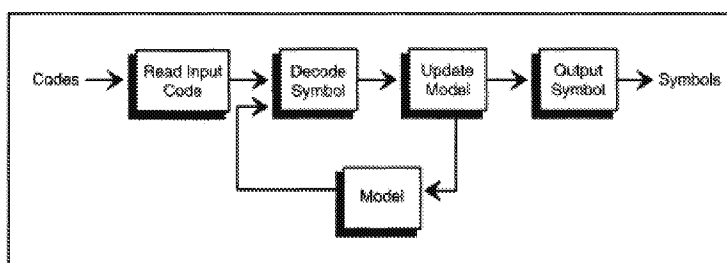


FIGURE 2.3 GENERAL ADAPTIVE DECOMPRESSION.

Id. at 19 (showing “Update Model” (state information) after encoding or decoding every piece of data). Nelson explains: “adaptive models start knowing essentially nothing about the data” so “when the program first starts it doesn’t do a very good job of compression.” *Id.* at 19. However, “Most adaptive algorithms tend to adjust quickly to the data stream and will begin turning in respectable compression ratios after only a few thousand bytes.” *Id.*

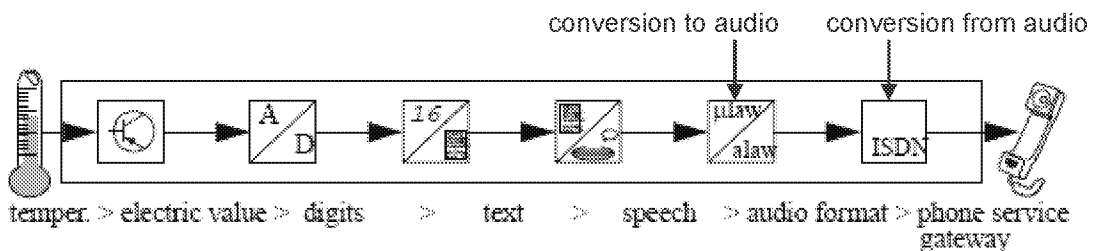
Thus, an obvious implementation of an adaptive algorithm would entail, for each packet, **retrieving** state information, using it to **perform** the compression or decompression processing, updating it to reflect the data in the most recent packet, and **storing** it so it can be applied to the next packet.

In view of their prominence and lossless nature, adaptive compression/decompression schemes were an obvious choice for any iPCSS converter component performing compression/decompression. Ex. 5 at 18 (“compression research in the last 10 years has concentrated on

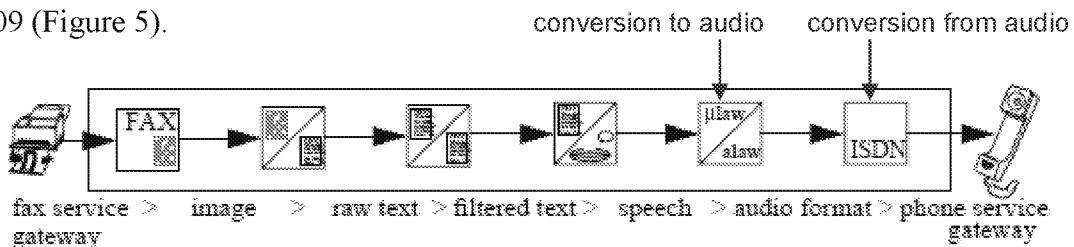
adaptive models”); Ex. A02 at 115 (“prohibit multiple lossy compression and decompression processes, if possible”).

As evident from Pfeifer96 Figure 3, any component converting to or from *audio* or *video* would have a settable compression parameter “c.” See Ex. A02 at 107 (Figure 3: “Generic conversion matrix”). Thus, it was obvious for any component converting to or from audio or video to read on the “state information” element of claim 1, by performing adaptive compression or decompression. Additionally, iPCSS-01 clearly envisions that multiple converters within the same chain may perform compression or decompression (just preferably not multiple *lossy* compression or decompression, and adaptive algorithms are not lossy). See *id.* at 115 (“prohibit multiple lossy compression and decompression processes, if possible”).

Thus, considering merely two converter chains expressly disclosed by Pfeifer96 in Figures 5 and 6, it is evident both chains have at least two components which convert to or from audio, and hence it was obvious that at least a plurality of their components would maintain “state information” as recited by claim 1, for each packet.



Id. at 109 (Figure 5).



Id. at 111 (Figure 6). Moreover, the last component in both chains is doubly stateful, because it would also maintain ISDN connection state, as explained above.

As another example, Pfeifer96 teaches that a user with only a “telephone” may wish to “attend” a “video conference.” *Id.* at 1. *See also, e.g., id.* at 15 (Figure 9: showing “speech” input and “video” output). A clearly obvious chain for connecting this call would comprise an ISDN adapter component for accepting an ISDN telephone call and converting it to audio, and a second converter for converting the audio to a video conference format. Clearly both converters would take audio as an input or output, hence it was obvious for every converter in this chain to maintain “state information” as recited by claim 1, for each packet. And again, the ISDN component is doubly stateful.

(c) Text-to-Speech Components

Pfeifer96 teaches a “Text-to-speech” (TTS) converter component. Pfeifer96 teaches that such TTS converters “analys[e] the grammatical structure of a sentence” in order to “improve the prosody of the speech, that is the intonation and phrase melody.” *Id.* at 8-9. Since the packet boundaries in an incoming stream of text will *not* generally be synchronized with the grammatical boundaries of sentences, it was obviously desirable for this component to buffer textual data across packets in order to obtain complete sentences whose “grammatical structure” could then be analyzed. *Id.* This would entail, for each packet, **retrieving** a previous incomplete sentence fragment (lacking an ending), **performing** the processing of the newly delivered textual data and as possible analyzing any complete sentences, and finally **storing** any remaining sentence fragment from the new data. Because a TTS component is included in both of the converter chains depicted in Figures 5 and 6 (analyzed above), this is yet another basis for finding it was obvious that at least a plurality of the components in those chains (and other

obvious chains incorporating a TTS component) would perform the “state information” elements recited by claim 1.

(d) Speech Recognition Components

Pfeifer96 teaches a component which performs “speech recognition” to convert “commands” and “dictation” to “text” (*e.g.*, for converting an incoming voice call to a stream of legible text on a user’s computer). *See* Ex. A02 at 110, 112, 107, 118. Pfeifer96 explains that such speech recognition software “can be speaker dependent,” but that “speaker adaptive” software is more “flexible.” *Id.* at 112. One of ordinary skill would recognize that for such a component to *adaptively* improve its recognition of a particular speaker’s voice, it would need to **retrieve** prior information about the speaker’s voice, **perform** the processing of recognizing speech using that information, and **store** updated information reflecting the speaker’s new utterance.

To summarize, it was obvious for many of the components disclosed by Pfeifer96 to maintain “state information” in the manner recited by claim 1 in order for them to function as described, and it was certainly clear that at least “a plurality of the components” in certain obvious chains would do so, for each packet.

(b) Claim 15

i. “demultiplexing packets of messages”

Claim 15 recites: “A method in a computer system demultiplexing packets of messages.” Under Implicit’s apparent claim constructions, Pfeifer96 renders obvious these elements. *See* Claim 1(i) (showing “A method in a computer system for processing a message having a sequence of packets”) above. Under Implicit’s apparent claim constructions, “demultiplexing” a packet is satisfied by routing a packet to the correct sequence of components for processing it—and it was obvious for Pfeifer96 to perform this function. *See* Section IV (“demultiplexing”)

above and Claim 1(iii) (showing “dynamically identifying a non-predefined sequence of components for processing the packets of the message”) above.

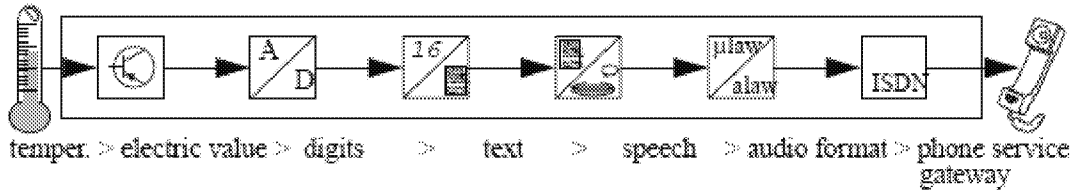
ii. “dynamically identifying a non-predefined sequence”

Claim 15 further recites “dynamically identifying a non-predefined sequence of components for processing each message based on the first packet of the message so that subsequent packets of the message can be processed without re-identifying the components, wherein dynamically identifying includes selecting individual components to create the non-predefined sequence of components after the first packet is received.” Under Implicit’s apparent claim constructions, Pfeifer96 renders obvious this element. *See* Claim 1(iii) (showing “for the first packet of the message, dynamically identifying a non-predefined sequence of components for processing the packets of the message”) and Claim 1(iv) (showing “storing an indication of each of the identified components so that the non-predefined sequence does not need to be re-identified for subsequent packets of the message”) above.

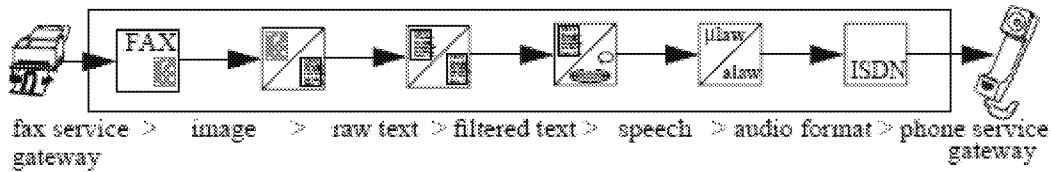
iii. “different . . . sequences of components can be identified”

Claim 15 further recites “different non-predefined sequences of components can be identified for different messages.” Under Implicit’s apparent claim constructions, Pfeifer96 renders obvious this element.

Since both the source and destination and the source and destination mediums will vary across messages, it is clear that correspondingly different sequences of converters will generally be required. *See, e.g.*, Ex. A02 at 109, 111. For example, a first communication may extend from a temperature sensor to an ISDN telephone:



Id. at 109 (Figure 5: “Converter chain for temperature to speech conversion with telephone delivery”). And a second communication may extend from a fax machine to an ISDN telephone, and it will therefore have a correspondingly different sequence of converters:



Id. at 111 (Figure 6: “Converter chain: fax reception, conversion to text and speech, telephone delivery”).

iv. “each component being a software routine”

Claim 15 further recites “each component being a software routine.” Under Implicit’s apparent claim constructions, Pfeifer96 renders obvious this element. *See* Claim 1(ii) above (showing “each component being a software routine”).

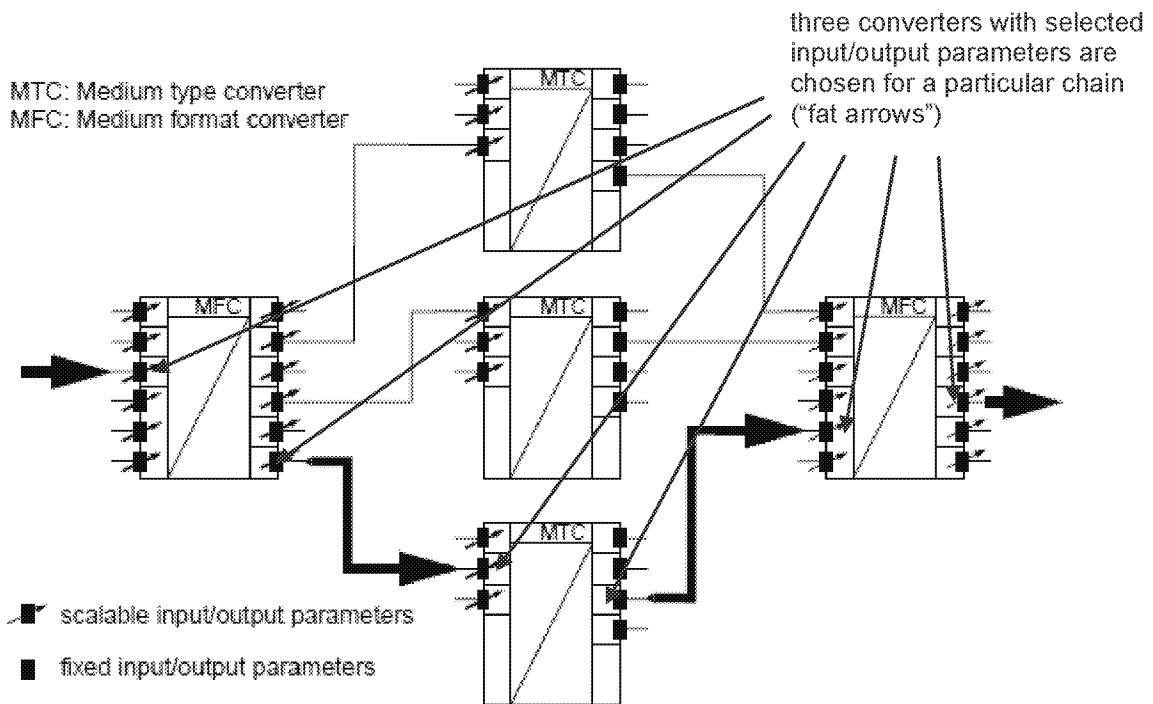
v. “selecting individual components”

Claim 15 further recites “dynamically identifying includes selecting individual components to create the non-predefined sequence of components.” Under Implicit’s apparent claim constructions, Pfeifer96 renders obvious these elements. *See* Claim 1(iii) above (showing “dynamically identifying includes selecting individual components to create the non-predefined sequence of components”).

vi. “state information”

Claim 15 finally recites “for each packet of each message, performing the processing of the identified non-predefined sequence of components of the message wherein state information generated by performing the processing of a component for a packet is available to the component when the component processes the next packet of the message.” Under Implicit’s apparent claim constructions, Pfeifer96 renders obvious this element.

As an initial matter, Pfeifer96 teaches that as part of its “framework of type and format converters” (illustrated in Figure 5), each component would maintain multiple “scalable input/output parameters” which would comprise (as recited by claim 1) “state information relating to the processing of the component” under Implicit’s apparent claim constructions. Ex. 8 at 108-09.



Id. at 108 (Figure 4: “Medium type conversion¹³ with format adaptation”). As explained above, such parameters are considered when performing the Quality of Service analysis comparing many possible chains that might be used. *See* Section iv above. Pfeifer96 teaches these parameters control, *e.g.*, “frame/sampling rate, quantization, resolution, size, color depth . . . compression technique,” and so on), and that the parameters chosen for the converters across any specific chain should be coordinated to avoid unnecessary loss of information. *See id.* at 107-08, 115, 124; Claim 1(iii) above. These parameters are “state information” which must be **stored** for each component, and which must be **retrieved** by each component in order to **perform** its processing: *e.g.*, in order to know which input or output “sampling rate” or “compression technique” to use when processing the packet. *See id.* It was obvious that all of the components in any particular chain would maintain such “parameters,” and therefore obvious that all of the components would satisfy the “state information” elements of claim 1.

Pfeifer96 also renders this element obvious from other perspectives.

For example, Pfeifer96 teaches that a user with only a “telephone” may wish to “attend” a “video conference.” *Id.* at 1. *See also, e.g., id.* at 15 (Figure 9: showing “speech” input and “video” output). A clearly obvious chain for connecting this call would comprise an ISDN adapter component for accepting an ISDN telephone call and converting it to audio, and a second converter for converting the audio to a video conference format. Clearly both converters would take audio as an input or output, hence it was obvious for every converter in this chain to maintain “state information” as recited by claim 1, for each packet. And again, the ISDN component would be doubly stateful. *See* Claim 1(v)(a) (“ISDN Adapter Components”) and (v)(b) (“Lossless Compression/Decompression”) above.

¹³ Medium *type* conversion (MTC) changes the format as well, since two different medium types cannot share the same format (*e.g.*, text vs. audible speech). *See* Section iii above.

As another example, Pfeifer teaches that conversions may be performed to convert “audio” to “video” and/or to “legible text” and/or to “Braille” (presumably for use by the blind). *See id.* at 107. Thus, a conversion clearly within the scope of Pfeifer96 would be converting an incoming ISDN voice call using a Speech Recognition component to produce video containing legible text (*e.g.*, for use by the deaf). *See id.* at 107, 111. As Pfeifer explains, “even conversions that sound strange in the first place might be of very practical relevance”—and this conversion is not at all strange or unexpected, particularly when viewing Figure 3. *Id.* at 109, 107. And clearly, it was obvious that every component in this obvious chain would maintain state in the manner recited by claim 15. The first component is an ISDN adapter, and it is doubly stateful because it must maintain ISDN connection state, and because it has an audio output. *See* Claim 1(v)(a) and (v)(b) above. The second and final component is triply stateful, because it accepts audio input, performs speech recognition, and emits video output. *See* Claim 1(v)(a), (v)(b), and (v)(d) (“Speech Recognition Components”) above.

(c) Claim 35

i. “instructions for demultiplexing packets of messages”

Claim 35 recites: “A computer-readable medium containing instructions for demultiplexing packets of messages.” Under Implicit’s apparent claim constructions, Pfeifer96 renders obvious this element. It was obvious that the software executed by the computer system of Pfeifer96 (including its sequences of software-based converters) would be stored on a “computer-readable medium.” *See* Claim 1(i) (showing “a computer system”) above.

ii. “dynamically identifying a . . . non-predefined sequence”

Claim 35 recites: “dynamically identifying a message-specific non-predefined sequence of components for processing the packets of each message upon receiving the first packet of the message.” Under Implicit’s apparent claim constructions, Pfeifer96 renders obvious this

element. *See* Claim 1(iii) (showing “for the first packet of the message, dynamically identifying a non-predefined sequence of components for processing the packets of the message”) above.

Pfeifer teaches that the “dynamically generated converter chain” is formed in the last of a series of steps which are triggered by the receipt of an incoming connection request. *See, e.g.*, Ex. A02 at 119 (Figure 10, including “configure media conversion” in “4th stage”), 122-24. And it is, of course, obvious that a connection request could comprise the first package of a message.

iii. “subsequent packets . . . can use the . . . sequence”

Claim 35 further recites that “subsequent packets of the message can use the message-specific non-predefined sequence identified when the first packet was received.” Under Implicit’s apparent claim constructions, Pfeifer96 renders obvious this element. *See* Claim 1(iv) (showing “storing an indication of each of the identified components so that the non-predefined sequence does not need to be re-identified for subsequent packets of the message”) above.

iv. “selecting individual components”

Claim 35 further recites that “dynamically identifying includes selecting individual components to create the message-specific non-predefined sequence of components.” Under Implicit’s apparent claim constructions, Pfeifer96 renders obvious this element. *See* Claim 1(iii) (showing “dynamically identifying includes selecting individual components to create the non-predefined sequence of components after the first packet is received”) above.

v. “state information”

Claim 35 further recites “for each packet of the message, invoking the identified non-predefined sequence of components in sequence to perform the processing of each component for the packet wherein each component saves message-specific state information so that that component can use the save message-specific state information when that component performs its processing on the next packet of the message.” Under Implicit’s apparent claim

constructions, Pfeifer96 renders obvious this element. *See* Claim 15(vi) (showing “for each packet of each message, performing the processing of the identified non-predefined sequence of components of the message wherein state information generated by performing the processing of a component for a packet is available to the component when the component processes the next packet of the message”) above.

3. Pfeifer96 in View of ISDN98 and Nelson Renders Obvious Claims 1, 15, and 35 Under § 103

The specification “ISDN Primary Rate User-Network Interface Specification” (Ex. 4, “ISDN98”) was published in August 1998 by Northern Telecom. The treatise “The Data Compression Book” (Ex. 5, “Nelson”) by Mark Nelson et al. was published on November 6, 1995. Neither was considered during the prosecution of the ‘163 patent.

If certain aspects recited in claims 1, 15, and 35 of the ‘163 patent are not deemed to be disclosed, inherent, or obvious over Pfeifer96 alone, then the inclusion of those aspects certainly would be obvious over Pfeifer96 in view of ISDN98 and Nelson, under 35 U.S.C. § 103.

ISDN98 and Nelson were cited above under MPEP § 2205 as confirming that certain information regarding ISDN (ISDN98) and compression (Nelson) would have been part of the standard background knowledge of those of ordinary skill in the art. *See* Section V.A.2 above.

It was also obvious to apply these references directly to Pfeifer96.

Pfeifer96 teaches that many of its specific converters would have a “parameter” specifying the “applied compression technique” employed by that converter. *E.g.*, Ex. A02 at 107. Nelson (“The Data Compression Book”) would have been an obvious place to look for information regarding the specific compression techniques that could be applied.

Pfeifer96 also teaches a converter for adapting audio data to ISDN. *E.g.*, Ex. A02 at 109. As the “ISDN Primary Rate User-Network Interface Specification,” ISDN98 would have been obvious place to look for information regarding what creating such an adapter would entail.

4. Pfeifer96 in View of Arbanowksi96 Renders Obvious Claims 1, 15, and 35 Under § 103

The dissertation “Generic Description of Telecommunication Services and Dynamic Resource Selection in Intelligent Communication Environments” by Stefan Arbanowski (Exhibit 11, “Arbanowksi96”) was published on October 6, 1996, and it was not considered during prosecution of the ‘163 patent.

If certain aspects recited in claims 1, 15, and 35 of the ‘163 patent are not deemed to be disclosed, inherent, or obvious over Pfeifer96, then the inclusion of those aspects certainly would be obvious over Pfeifer96 in view of Arbanowksi96, under 35 U.S.C. § 103.

It was obvious to apply Arbanowksi96 to Pfeifer96 because both documents describe the “Intelligent Personal Communication Support System (iPCSS),” with Arbanowksi96 presenting further detail regarding, *e.g.*, the dynamic selection of converter chains. *See, e.g.*, Ex. 11 at 8, 13-14, 44-54 (“Dynamic Resource Selection”).

Because Pfeifer96 and Arbanowksi96 are so similar in approach and detail, Arbanowksi96 reinforces and confirms the analysis presented in Section V.A.2 (Pfeifer96 103) in many ways. Some exemplary aspects of Arbanowksi96 are pointed out particularly below.

(a) Claim 1

i. “A method in a computer system . . .”

Claim 1 recites: “A method in a computer system for processing a message having a sequence of packets, the method comprising” Under Implicit’s apparent claim constructions, Pfeifer96 in view of Arbanowksi96 renders obvious this element. Arbanowksi96

confirms a “computer system” would be used. *See, e.g.*, Ex. 11 at 6-8, 19 (“Computational Objects”).

Claim 1 further recites the method is “for processing a message having a sequence of packets.” Arbanowksi96 discloses that the “bearer” (which is “the physical network” connected to an iPCSS “service gateway” can consist of the following network types, virtually all of which are inherently packet-oriented:

<i>possible values</i>	<i>short description</i>
ATM	Asynchronous Transfer Mode [ITU-T I.361]
FDDI	Fiber Distributed Data Interface [ITU-T 3914x]
ISDN	Integrated Service Digital Network [ITU-T I.320]
B-ISDN	Broadband Integrated Service Digital Network [ITU-T I.321]
DQDB	Distributed Queue Dual Bus [IEEE 802.6]
Ethernet	normal 10Mbit Ethernet [IEEE 802.3]
GSM	Global System for Mobile Communication
DCS-1800	Digital Cellular System
PSTN	Public Switched Telephone Network

Id. at 33 (Table 3-6: “Possible Values for the Attribute Bearer”). Thus, an incoming communication over any of those packet-oriented networks would comprise “a message having a sequence of packets.”

ii. “a plurality of components”

Claim 1 further recites: “providing a plurality of components, each component being a software routine for converting data with an input format into data with an output format.”

Under Implicit’s apparent claim constructions, Pfeifer96 in view of Arbanowksi96 renders obvious these elements. *See, e.g., id.* at 9 (“iPCSS – Theory of Conversion”), 50 (“The example

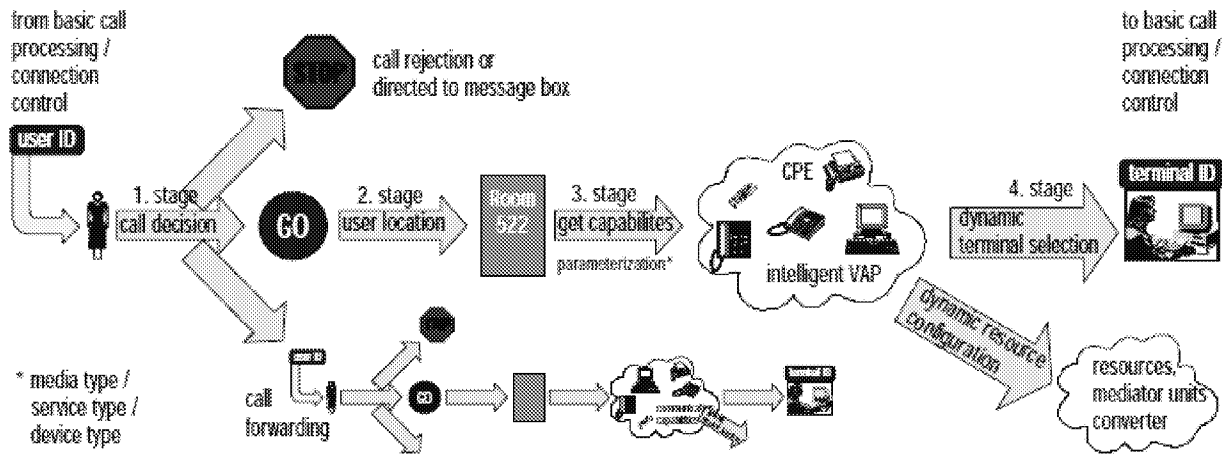
demonstrates a scenario with three converters and four terminals to lead into 60 possible converter chains.”).

iii. “*dynamically identifying a non-predefined sequence*”

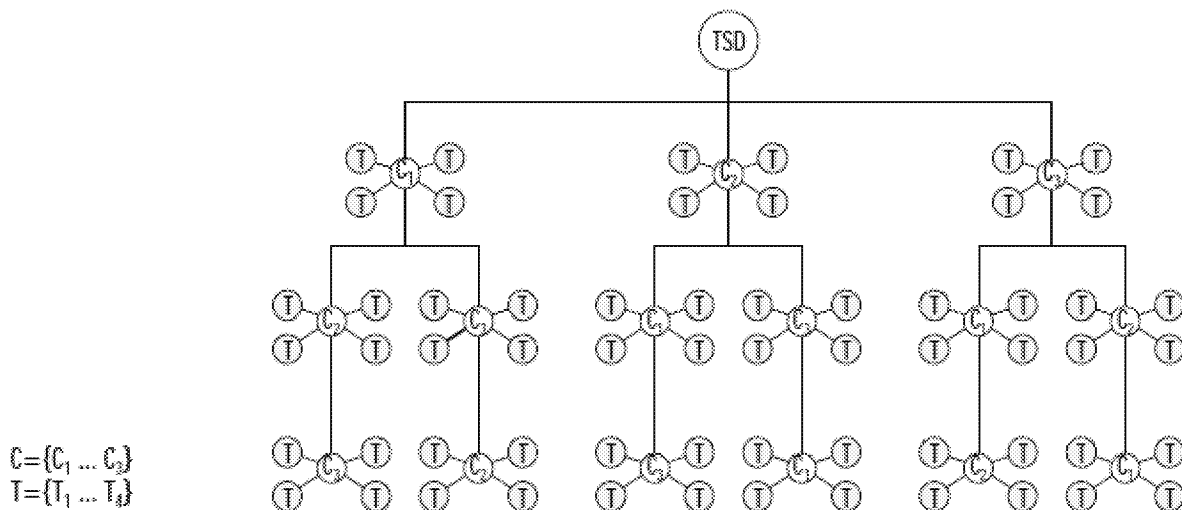
Claim 1 further recites: “for the first packet of the message, dynamically identifying a non-predefined sequence of components for processing the packets of the message such that the output format of the components of the non-predefined sequence match the input format of the next component in the non-predefined sequence, wherein dynamically identifying includes selecting individual components to create the non-predefined sequence of components after the first packet is received.” Under Implicit’s apparent claim constructions, Pfeifer96 in view of Arbanowksi96 renders obvious this element.

As suggested by its title (“Generic Description of Telecommunication Services and *Dynamic Resource Selection* in Intelligent Communication Environments”), Arbanowksi96 confirms that the chain of converters for connecting a call is identified dynamically and is not predefined.

Like Pfeifer96, Arbanowksi96 discloses a four-stage call connection process wherein the selection of a terminal in the user’s vicinity and the dynamic configuration of a chain of converters to that terminal occurs in the final stage. *E.g., id.* at 6-7, 13-15, 49 (“Finding a Matching Device”), 50 (“Finding a Possible Chain”).



Id. at 14 (Figure 2-9: “iPCSS – Call Handling”). Arbanowksi96 also provides additional detail on the elaborate process by which many possible chains of converters are considered in the course of initially connecting a call to one of the terminals in the called user’s current vicinity. *E.g., id.* at 44 (the chosen terminal should not be “idle” and not “busy” or “down”), 49 (“Finding a Matching Device”), 50 (“Finding a Possible Chain”: “The example demonstrates a scenario with three converters and four terminals to lead into 60 possible converter chains”), 52 (“Calculating the most appropriate chain”)



Id. at 51 (Figure 4-8: “Dynamic Resource Selection – Possible Converter Chains”). Because iPCSS constructs many possible chains of individual converters and performs a complex “Quality of Service” analysis on the possible chains in the course of connecting a call (*i.e.*, after it has received the first packet of the message), it is clear that the ultimately selected chain of converters is selected “dynamically” and is not “predefined.” *See, e.g., id.* at 49-54.

Claim 1 further recites that the “dynamically identifying includes selecting individual components to create the non-predefined sequence of components after the first packet is received.” Arbanowski⁹⁶ makes clear that iPCSS considers the concatenation of many possible individual converters in the course of connecting a call, and that the ultimately selected chain will have been composed by selecting individual components. *E.g., id.* at 50 (“Finding a Possible Chain”: “The example demonstrates a scenario with three converters and four terminals to lead into 60 possible converter chains”).

iv. “storing an indication of . . . the identified components”

Claim 1 further recites “storing an indication of each of the identified components so that the non-predefined sequence does not need to be re-identified for subsequent packets of the message.” Under Implicit’s apparent claim constructions, Pfeifer⁹⁶ in view of Arbanowski⁹⁶ renders obvious this element. Arbanowski⁹⁶ greatly elaborates on the process by which a “dynamically generated converter chain” is selected for connecting an incoming call. *See Ex. A02 at 125; Ex. 11 at 44-54 (“Dynamic Resource Selection”).* The “dynamically generated converter chain” of Pfeifer⁹⁶ is generated only once for an incoming communication and it is used to process the entire communication. *See Ex. A02 at 122-27. See also Section V.A.2 (Pfeifer⁹⁶ 103) at Claim 1(vi).*

v. “state information”

Claim 1 further recites “for each of a plurality of packets of the message in sequence, for each of a plurality of components in the identified non-predefined sequence, **retrieving** state information relating to performing the processing of the component with the previous packet of the message; **performing** the processing of the identified component with the packet and the retrieved state information; and **storing** state information relating to the processing of the component with the packet for use when processing the next packet of the message.” Under Implicit’s apparent claim constructions, Pfeifer96 in view of Arbanowksi96 renders obvious this “state information” elements. Arbanowksi96 provides additional detail on the types of conversions and converter chains that might be assembled (*e.g.*, Ex. 11 at 28-38), and additional detail on the process by which the “scalable input/output parameters” of the converter components in Pfeifer96 (which are “state information”) would be configured (*e.g.*, *id.* at 50-54). It thus confirms the analysis presented under Section V.A.2 (Pfeifer96 103) in several ways. *See* Section V.A.2 (Pfeifer96 103) at Claim 1(v).

(b) Claim 15

i. “demultiplexing packets of messages”

Claim 15 recites: “A method in a computer system demultiplexing packets of messages.” Under Implicit’s apparent claim constructions, Pfeifer96 in view of Arbanowksi96 renders obvious this element. *See* Claim 1(i) (showing “A method in a computer system for processing a message having a sequence of packets”) above.

ii. “dynamically identifying a non-predefined sequence”

Claim 15 further recites “dynamically identifying a non-predefined sequence of components for processing each message based on the first packet of the message so that subsequent packets of the message can be processed without re-identifying the components.” Under Implicit’s apparent claim constructions, Pfeifer96 in view of Arbanowksi96 renders

obvious these elements. *See* Claim 1(iii) (showing “for the first packet of the message, dynamically identifying a non-predefined sequence of components for processing the packets of the message”) and Claim 1(iv) (showing “storing an indication of each of the identified components so that the non-predefined sequence does not need to be re-identified for subsequent packets of the message”) above.

iii. “different . . . sequences of components can be identified”

Claim 15 further recites “different non-predefined sequences of components can be identified for different messages.” Under Implicit’s apparent claim constructions, Pfeifer⁹⁶ in view of Arbanowski⁹⁶ renders obvious this element. *See, e.g.*, Ex. 11 at 50 (“Finding a Possible Chain”: “The example demonstrates a scenario with three converters and four terminals to lead into 60 possible converter chains”). *See also* Claim 1(iii) above.

iv. “each component being a software routine”

Claim 15 further recites “each component being a software routine.” Under Implicit’s apparent claim constructions, Pfeifer⁹⁶ in view of Arbanowski⁹⁶ renders obvious this element. *See* Claim 1(ii) above (showing “each component being a software routine”).

v. “selecting individual components”

Claim 15 further recites “dynamically identifying includes selecting individual components to create the non-predefined sequence of components.” Under Implicit’s apparent claim constructions, Pfeifer⁹⁶ in view of Arbanowski⁹⁶ renders obvious this element. *See* Claim 1(iv) (showing “dynamically identifying includes selecting individual components to create the non-predefined sequence of components”) above.

vi. “state information”

Claim 15 finally recites “for each packet of each message, performing the processing of the identified non-predefined sequence of components of the message wherein state information

generated by performing the processing of a component for a packet is available to the component when the component processes the next packet of the message.” Under Implicit’s apparent claim constructions, Pfeifer96 in view of Arbanowksi96 renders obvious this element. *See* Claim 1(v) above (showing similar element).

(c) Claim 35

i. “instructions for demultiplexing packets of messages”

Claim 35 recites: “A computer-readable medium containing instructions for demultiplexing packets of messages.” Under Implicit’s apparent claim constructions, Pfeifer96 in view of Arbanowksi96 renders obvious this element. *See* Claim 1(i) above (showing “a computer system”).

ii. “dynamically identifying a . . . non-predefined sequence”

Claim 35 recites: “dynamically identifying a message-specific non-predefined sequence of components for processing the packets of each message upon receiving the first packet of the message.” Under Implicit’s apparent claim constructions, Pfeifer96 in view of Arbanowksi96 renders obvious this element. *See* Claim 1(iii) (showing “for the first packet of the message, dynamically identifying a non-predefined sequence of components for processing the packets of the message”) above.

iii. “subsequent packets . . . can use the . . . sequence”

Claim 35 further recites that “subsequent packets of the message can use the message-specific non-predefined sequence identified when the first packet was received.” Under Implicit’s apparent claim constructions, Pfeifer96 in view of Arbanowksi96 renders obvious this element. *See* Claim 1(iv) (showing “storing an indication of each of the identified components so that the non-predefined sequence does not need to be re-identified for subsequent packets of the message”) above.

iv. “*selecting individual components*”

Claim 35 further recites that “dynamically identifying includes selecting individual components to create the message-specific non-predefined sequence of components.” Under Implicit’s apparent claim constructions, Pfeifer96 in view of Arbanowksi96 renders obvious this element. *See* Claim 1(iii) (showing “dynamically identifying includes selecting individual components to create the non-predefined sequence of components after the first packet is received”) above.

v. “*state information*”

Claim 35 further recites “for each packet of the message, invoking the identified non-predefined sequence of components in sequence to perform the processing of each component for the packet wherein each component saves message-specific state information so that that component can use the save message-specific state information when that component performs its processing on the next packet of the message.” Under Implicit’s apparent claim constructions, Pfeifer96 in view of Arbanowksi96 renders obvious this element. *See* Claim 1(v) (showing similar element) above.

5. Pfeifer96 in View of Pfeifer97 Renders Obvious Claims 1, 15, and 35 Under § 103

The article “Resource Selection in Heterogeneous Communication Environments using the Teleservice Descriptor” by Tom Pfeifer, Stefan Arbanowski, and Radu Popescu-Zeletin (Exhibit 12, “Pfeifer97”) was published by December 19, 1997, and it was not considered during prosecution of the ‘163 patent.

If certain aspects recited in claims 1, 15, and 35 of the ‘163 patent are not deemed to be disclosed, inherent, or obvious over Pfeifer96, then the inclusion of those aspects certainly would be obvious over Pfeifer96 in view of Pfeifer97, under 35 U.S.C. § 103.

It was obvious to apply Pfeifer97 to Pfeifer96 because both documents describe the “Intelligent Personal Communication Support System (iPCSS),” with Pfeifer97 presenting further detail regarding, *e.g.*, the dynamic selection of converter chains. *See, e.g.*, Ex. 12 at 132, 143-50.

Because Pfeifer96 and Pfeifer97 are so similar in approach and detail, Pfeifer97 reinforces and confirms the analysis presented in Section V.A.2 (Pfeifer96 103) in many ways. Some exemplary aspects of Pfeifer97 are pointed out particularly below.

(a) *Claim 1*

i. *“a computer system”*

Claim 1 recites: “A method in a computer system for processing a message having a sequence of packets, the method comprising” Under Implicit’s apparent claim constructions, Pfeifer96 in view of Pfeifer97 renders obvious this elements. Pfeifer97 confirms a “computer system” would be used. *See, e.g.*, Ex. 12 at 150-52 (“Computational Model”).

Claim 1 further recites the method is “for processing a message having a sequence of packets.” Pfeifer97 discloses that the “bearer” (which is “the physical network” connected to an iPCSS “service gateway” can consist of the following network types, virtually all of which are inherently packet-oriented:

possible values	short description
ATM	Asynchronous Transfer Mode (ITU-T I.361)
FDDI	Fibre Distributed Data Interface (ITU-T 3914x)
ISDN	Integrated Service Digital Network (ITU-T I.320)
B-ISDN	Broadband Integrated Service Digital Network (ITU-T I.321)
DQDB	Distributed Queue Dual Bus (IEEE 802.6)
Ethernet	normal 10Mbit Ethernet (IEEE 802.3)
GSM	Global System for Mobile Communication
DCS-1800	Digital Cellular System
PSTN	Public Switched Telephone Network

Id. at 137-38 (Table 6: “Possible Values for the Attribute Bearer”). Thus, an incoming communication over any of those packet-oriented networks would comprise “a message having a sequence of packets.”

ii. “a plurality of components”

Claim 1 further recites: “providing a plurality of components, each component being a software routine for converting data with an input format into data with an output format.”

Under Implicit’s apparent claim constructions, Pfeifer96 in view of Pfeifer97 renders obvious this element. *See, e.g., id.* at 136-37 (“Conversion”: “Some conversions are only possible by using more than one conversion steps”), 148-49 (“Finding a Possible Converter Chain”).

iii. “dynamically identifying a non-predefined sequence”

Claim 1 further recites: “for the first packet of the message, dynamically identifying a non-predefined sequence of components for processing the packets of the message such that the output format of the components of the non-predefined sequence match the input format of the next component in the non-predefined sequence, wherein dynamically identifying includes selecting individual components to create the non-predefined sequence of components after the first packet is received.” Under Implicit’s apparent claim constructions, Pfeifer96 in view of Pfeifer97 renders obvious this element.

Pfeifer97 confirms that the chain of converters for connecting a call is identified dynamically and is not predefined. *See, e.g., id.* at 146 (“Dynamic Resource Selection”),

Like Pfeifer96, Pfeifer97 discloses a four-stage call connection process wherein the selection of a terminal in the user’s vicinity and the dynamic configuration of a chain of converters to that terminal occurs in the final stage. *E.g., id.* at 137, 143-45 (“Automatic Resource Selection” which “find[s] the most appropriate terminal . . . at the [called] user’s current location dynamically”).

Pfeifer97 also provides additional detail on the elaborate process by which many possible chains of converters are considered in the course of initially connecting a call to one of the terminals in the called user's current vicinity. *E.g., id.* at 145 (the chosen terminal should not be "idle" and not "busy" or "down"), 147 ("Calculating the Most Appropriate Device"), 148 ("Finding a Possible Converter Chain": "Five converters and five terminals allow 600 combinations, and one hundred converters and terminals lead to over 500.000 theoretical possibilities The main task of the algorithm is therefore to reduce the space of search.").

Because iPCSS constructs many possible chains of individual converters and performs a complex "Quality of Service" analysis on the possible chains in the course of connecting a call (*i.e.*, after it has received the first packet of the message), it is clear that the ultimately selected chain of converters is selected "dynamically" and is not "predefined." *See, e.g., id.* at 145-50.

Claim 1 further recites that the "dynamically identifying includes selecting individual components to create the non-predefined sequence of components after the first packet is received." Under Implicit's apparent claim constructions, Pfeifer96 in view of Pfeifer97 renders obvious this element. It is clear that iPCSS considers the concatenation of many possible individual converters in the course of connecting a call, and that the ultimately selected chain will have been composed by selecting individual components. *E.g., id.* at 148-49 ("Finding a Possible Converter Chain": "Five converters and five terminals allow 600 combinations, and one hundred converters and terminals lead to over 500.000 theoretical possibilities").

iv. "storing an indication of . . . the identified components"

Claim 1 further recites "storing an indication of each of the identified components so that the non-predefined sequence does not need to be re-identified for subsequent packets of the message." Under Implicit's apparent claim constructions, Pfeifer96 in view of Pfeifer97 renders obvious this element. Pfeifer97 greatly elaborates on the process by which a "dynamically

generated converter chain” is selected for connecting an incoming call. *See* Ex. 3 at 124-25, Ex. 12 at 143-153 (“Automatic Resource Selection”). The “dynamically generated converter chain” of Pfeifer96 is generated only once for an incoming communication and it is used to process the entire communication. *See* Ex. 3 at 122-27. *See also* Section V.A.2 (Pfeifer96 103) at Claim 1(iv).

v. “state information”

Claim 1 further recites “for each of a plurality of packets of the message in sequence, for each of a plurality of components in the identified non-predefined sequence, **retrieving** state information relating to performing the processing of the component with the previous packet of the message; **performing** the processing of the identified component with the packet and the retrieved state information; and **storing** state information relating to the processing of the component with the packet for use when processing the next packet of the message.” Under Implicit’s apparent claim constructions, Pfeifer96 in view of Pfeifer97 renders obvious this “state information” element. Pfeifer97 provides additional detail on the types of conversions and converter chains that might be assembled (*e.g.*, Ex. 12 at 133-37, 143-44), and additional detail on the process by which the “scalable input/output parameters” of the converter components in Pfeifer96 (which are “state information”) would be configured (*e.g.*, *id.* at 144-45, 149-50). It thus confirms the analysis presented under Section V.A.2 (Pfeifer96 103) in several ways. *See* Section V.A.2 (Pfeifer96 103) at Claim 1(v).

(b) Claim 15

i. “demultiplexing packets of messages”

Claim 15 recites: “A method in a computer system demultiplexing packets of messages.” Under Implicit’s apparent claim constructions, Pfeifer96 in view of Pfeifer97 renders obvious

this element. *See* Claim 1(i) above (showing “A method in a computer system for processing a message having a sequence of packets”).

ii. “dynamically identifying a non-predefined sequence”

Claim 15 further recites “dynamically identifying a non-predefined sequence of components for processing each message based on the first packet of the message so that subsequent packets of the message can be processed without re-identifying the components.” Under Implicit’s apparent claim constructions, Pfeifer96 in view of Pfeifer97 renders obvious this element. *See* Claim 1(iii) (showing “for the first packet of the message, dynamically identifying a non-predefined sequence of components for processing the packets of the message”) and Claim 1(iv) (showing “storing an indication of each of the identified components so that the non-predefined sequence does not need to be re-identified for subsequent packets of the message”) above.

iii. “different . . . sequences of components can be identified”

Claim 15 further recites “different non-predefined sequences of components can be identified for different messages.” Under Implicit’s apparent claim constructions, Pfeifer96 in view of Pfeifer97 renders obvious this element. *See, e.g.*, Ex. 12 at 19 (“Finding a Possible Converter Chain”: “Five converters and five terminals allow 600 combinations, and one hundred converters and terminals lead to over 500.000 theoretical possibilities”). *See also* Claim 1(iii) above.

iv. “each component being a software routine”

Claim 15 further recites “each component being a software routine.” Under Implicit’s apparent claim constructions, Pfeifer96 in view of Pfeifer97 renders obvious this element. *See* Claim 1(ii) above (showing “each component being a software routine”).

v. “selecting individual components”

Claim 15 further recites “dynamically identifying includes selecting individual components to create the non-predefined sequence of components.” Under Implicit’s apparent claim constructions, Pfeifer96 in view of Pfeifer97 renders obvious this element. *See* Claim 1(iii) above (showing “dynamically identifying includes selecting individual components to create the non-predefined sequence of components”).

vi. “state information”

Claim 15 finally recites “for each packet of each message, performing the processing of the identified non-predefined sequence of components of the message wherein state information generated by performing the processing of a component for a packet is available to the component when the component processes the next packet of the message.” Under Implicit’s apparent claim constructions, Pfeifer96 in view of Pfeifer97 renders obvious this element. *See* Claim 1(v) above (showing similar element).

(c) Claim 35

i. “instructions for demultiplexing packets of messages”

Claim 35 recites: “A computer-readable medium containing instructions for demultiplexing packets of messages.” Under Implicit’s apparent claim constructions, Pfeifer96 in view of Pfeifer97 renders obvious this element. *See* Claim 1(i) above.

ii. “dynamically identifying a . . . non-predefined sequence”

Claim 35 recites: “dynamically identifying a message-specific non-predefined sequence of components for processing the packets of each message upon receiving the first packet of the message.” Under Implicit’s apparent claim constructions, Pfeifer96 in view of Pfeifer97 renders obvious this element. *See* Claim 1(iii) (showing “for the first packet of the message, dynamically identifying a non-predefined sequence of components for processing the packets of the message”) above.

iii. *“subsequent packets . . . can use the . . . sequence”*

Claim 35 further recites that “subsequent packets of the message can use the message-specific non-predefined sequence identified when the first packet was received.” Under Implicit’s apparent claim constructions, Pfeifer96 in view of Pfeifer97 renders obvious this element. *See* Claim 1(vi) (showing “storing an indication of each of the identified components so that the non-predefined sequence does not need to be re-identified for subsequent packets of the message”) above.

iv. *“selecting individual components”*

Claim 35 further recites that “dynamically identifying includes selecting individual components to create the message-specific non-predefined sequence of components.” Under Implicit’s apparent claim constructions, Pfeifer96 in view of Pfeifer97 renders obvious this element. *See* Claim 1(iii) (showing “dynamically identifying includes selecting individual components to create the non-predefined sequence of components after the first packet is received”) above.

v. *“state information”*

Claim 35 further recites “for each packet of the message, invoking the identified non-predefined sequence of components in sequence to perform the processing of each component for the packet wherein each component saves message-specific state information so that that component can use the save message-specific state information when that component performs its processing on the next packet of the message.” Under Implicit’s apparent claim constructions, Pfeifer96 in view of Pfeifer97 renders obvious this element. *See* Claim 15(vi) above (showing “for each packet of each message, performing the processing of the identified non-predefined sequence of components of the message wherein state information generated by

performing the processing of a component for a packet is available to the component when the component processes the next packet of the message”).

6. Pfeifer96 in View of Cox Renders Obvious Claims 1, 15, and 35 Under § 103

The treatise “Superdistribution: Objects as Property on the Electronic Frontier” by Brad Cox (Exhibit P04, “Cox”) was published on June 28, 1996. It was not considered during prosecution of the ‘163 patent.

If certain aspects recited in claims 1, 15, and 35 of the ‘163 patent are not deemed to be disclosed, inherent, or obvious over Pfeifer96, then the inclusion of those aspects certainly would be obvious over Pfeifer96 in view of Cox, under 35 U.S.C. § 103.

(a) Claim 1

i. “state information”

Claim 1 recites in part: “for each of a plurality of packets of the message in sequence, for each of a plurality of components in the identified non-predefined sequence, **retrieving** state information relating to performing the processing of the component with the previous packet of the message; **performing** the processing of the identified component with the packet and the retrieved state information; and **storing** state information relating to the processing of the component with the packet for use when processing the next packet of the message.” Under Implicit’s apparent claim constructions, Pfeifer96 in view of Cox renders obvious this element.

The basic thesis of Cox is that “the software development industry” is in the midst of a “Software Crisis” caused by its over-reliance on a “pay-per-copy” revenue model. *See, e.g.*, Ex. P04 at x-xi, 45-73 (Chapter 3: “Software Crisis”), 143-165 (Chapter 6: “Out of the Crisis”).

A major aspect of this crisis is that “small-granularity, reusable software components” are essential for building complex systems in an efficient manner, yet it is precisely such small

components which lack a suitable revenue model. *See, e.g., id.* at 51-53 (“Software Complexity,” lamenting that “software is hand-crafted, fabricated from first principles and not assembled from prefabricated components”), 153 (“We continue to fabricate everything from first principles because there is no reliable way for those who might build smaller components to get paid. No one invests in fabricating small software for others to assemble because the low-tech revenue protection schemes do not work for small-granularity, low-priced objects such as reusable software components.”).

Cox illustrates this lack of suitable revenue model for small, reusable components with a telling example. Suppose “Vendor E” sells one of the smallest software components imaginable: a “string compare component.” *Id.* at 153, 145.

Vendor E's best current option is to attach his small-granularity product to something much larger, such as . . . [an off-the-shelf compiler product]. The string compare component is then perceived as free by the customer and by the vendor as a cost center, not a profit center. This almost guarantees that managers and stockholders will see **inadequate incentives to test, document, and maintain reusable components** to the point that others will be prepared to reuse them. The other option, which is somewhat **less feasible**, is to bundle the string compare routine with a large number of other components to produce a library large enough to be worth the trouble of marketing it. The accepted model for selling such libraries today is to charge a single relatively large fee, typically in the \$500-\$5000 range, for a license that allows the customer to include the library in larger applications.

This leads directly to the **debilitating consequences** I discussed in Chapter 2. Since the price is large and not proportional to utility, the vendor's income arrives all at once at the very beginning of the relationship with the customer. This leads to precisely the same **dysfunction** that farmers and millers would suffer if the miller sold the baker a license to replicate all the wheat and flour he might ever need in advance. Since the fee is large and fixed, small bakers couldn't afford it and large bakers would have an unfair advantage. Worse yet, the miller would have **no incentive to improve the product over time**.

Id. at 153-54 (emphasis added). *See also id.* at 31-33 (a pertinent section of “Chapter 2”).

The solution to this crisis, according to Cox, is “an invocation-based metering” approach which Cox styles “Superdistribution.” *See, e.g., id.* at 155, 169 (“*invocation-based revenue collection*”) (emphasis in original). The goal of this “Superdistribution” approach is to “provide a **meter** that supports revenue collection for **components of any granularity**.” *Id.* at 156 (emphasis added). Assessing royalties based on *actual usage* of a component would solve a number of problems, including the problem of Vendor E:

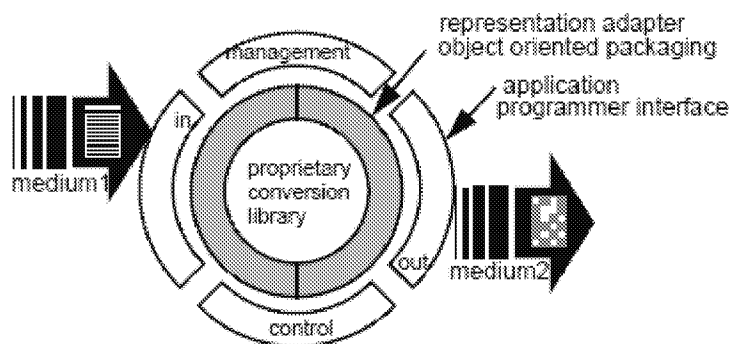
Instead of paying a large fee up-front, all customers, large and small, get the component for free. Later, when they begin to sell their own products based on this component, they pay a negotiated fee for using their subvendor's product. The subvendor now receives a continuing revenue stream that is directly proportional to the utility his component provides to his customers.

Id. at 154.

Thus, central to this “Superdistribution” approach advocated by the book is this cumulative “invocation-counting” mechanism which “merely collects information about invocations”: *i.e.*, each time a software component employing this system is invoked, its usage meter is incremented. *See, e.g., id.* at 174-75, 178. A “financial institution” would later obtain these “invocation counts” and convert them “to financial amounts due.” *Id.* at 182.

As suggested by its title (“Superdistribution”), this entire book is devoted to proposing this “invocation-based metering” approach it styles “Superdistribution,” and explaining the need for it. *See, e.g., id.* at 155 (“Superdistribution” section in chapter entitled “Out of the Crisis”), 183 (“Everything is based on the simple invocation-metering logic discussed earlier”). Thus, the excerpts cited above capture only a small portion of the extensive case made by the book for this solution.

In any event, upon reading Cox, one of ordinary skill in the art could not fail to see its relevance to the small, reusable converter components of Pfeifer96. Though the Cox approach could obviously be applied to metering the usage of *any* components in a large software system such as iPCSS, the converter components of Pfeifer96 in particular would stand out as especially likely candidates for this treatment, because Pfeifer96 *expressly teaches* they may be “proprietary” external components obtained “from different manufacturers,” rather than components developed purely internally. *See* Ex. A02 (Pfeifer96) at 108, 113-14.



Id. (Pfeifer96) at 113 (Figure 7: “Generic converter model,” showing a “proprietary conversion library”).

These converters are, moreover, the quintessence of the “small-granularity reusable software components” discussed throughout Cox. *E.g.*, Ex. P04 at 15, 35, 71, 90, 145, 153. Indeed, Pfeifer96 teaches an entire infrastructure for mixing and matching these components interchangeably into “converter chain[s]” for various purposes—so they are continually being reused on even a message-by-message basis. *E.g.*, Ex. A02 at 116, 122-24.

Motivation to apply the “Superdistribution” technique to the iPCSS converters is supplied throughout Cox, including fostering “a commercially robust market in prefabricated software components.” *See* Ex. P04 at ix, 143, 163-64. Such a market would make it easier for the iPCSS provider to quickly obtain high-quality converters to bridge between continuously emerging

communications formats. *See id.* Or to put this more clearly from the supplier’s perspective, unless the “Superdistribution” technique of “invocation-based metering” was supported by iPCSS, would-be vendors of such proprietary conversion libraries would experience precisely the economic conundrum that confronted Vendor E above, and thus few such libraries would be offered for sale to iPCSS. *See id.* at 153-55.

Having determined to apply the “Superdistribution” technique to at least to the iPCSS converter system, one of ordinary skill would recognize that as a general matter, a separate invocation count must be maintained for each converter component. This is, simply, the Cox technique: any distinct product being billed with the technique (such as a particular converter component) requires its own invocation count. *See, e.g.,* Ex. P04 (Cox) at 174-76. For example, each converter product might be from a different vendor who needs to be separately paid, and even if several converters were supplied by the same vendor, they may be priced differently (*e.g.,* depending on their complexity). *See, e.g.,* Ex. A02 (Pfeifer96) at 109 (“The range of conversions varies tremendously in effort, cost, and required resources. Some kinds are easy to implement with two lines of C code . . . while others are highly complex, requiring . . . approaches of artificial intelligence (*e.g.,* speech recognition)”). Though conceivably the iPCSS could choose not to apply the technique *at all* to a few low-value converters (*e.g.,* if they are available for free), it is obvious that *any or all* of the converters in any particular chain of converters could employ the Cox technique.

Claim 1 recites in part: “for each of a plurality of packets of the message in sequence, for each of a plurality of components in the identified non-predefined sequence, **retrieving** state information relating to performing the processing of the component with the previous packet of the message; **performing** the processing of the identified component with the packet and the

retrieved state information; and **storing** state information relating to the processing of the component with the packet for use when processing the next packet of the message.”

This element would clearly be satisfied by maintaining a cumulative invocation count for each converter. A component’s invocation count is plainly “state information relating to the processing of the component”: indeed, it incremented every time the component processes another packet. Moreover, maintaining its invocation count would be an essential *part* of the processing **performed** by the component, since without it, the entire economic model fails, iPCSS would have no legal right to use the component, and it would have to be removed. Indeed, the entire system of Cox is premised on the ability of these components to update their invocation counts as an integral part of their processing. *See, e.g.*, Ex. P04 at 174-76. And finally, of course, maintaining a cumulative invocation count would clearly entail: (1) **retrieving** the previous count each time the component is invoked (*e.g.*, to process another packet), (2) incrementing the previous count, and (3) **storing** the updated result.

As discussed above, it was obvious that any of all of the converters in a chain would employ the Cox technique. Thus, although claim 1 recites “for each of a *plurality* of packets of the message in sequence, for each of a *plurality* of components in the . . . sequence,” Pfeifer96 in view of Cox renders obvious that all of the above operations would be performed for *each* packet and for *each* component.

(b) *Claim 15*

i. “*state information*”

Claim 15 recites in part: “for each packet of each message, performing the processing of the identified non-predefined sequence of components of the message wherein state information generated by performing the processing of a component for a packet is available to the

component when the component processes the next packet of the message.” Under Implicit’s apparent claim constructions, Pfeifer96 in view of Cox renders obvious this element.

As explained above under Claim 1, Pfeifer96 in view of Cox renders obvious that each component would generate state information (in form of a cumulative invocation count) which was available to the component when the component process the next packet of the message (so that it could increment the cumulative invocation count yet again).

(c) Claim 35

Claim 35 recites in part: “for each packet of the message, invoking the identified non-predefined sequence of components in sequence to perform the processing of each component for the packet wherein each component saves message-specific state information so that that component can use the save message-specific state information when that component performs its processing on the next packet of the message.” Under Implicit’s apparent claim constructions, Pfeifer96 in view of Cox renders obvious this element.

As explained above under Claim 1, Pfeifer96 in view of Cox renders obvious that each component would generate state information (in form of a cumulative invocation count) which was available to the component when the component process the next packet of the message (so that it could increment the cumulative invocation count yet again). This cumulative count is message-specific in sense that, *e.g.*, it would clearly reflect the number of packets in the particular message.

7. Pfeifer96 in View of Meer96 Renders Obvious Claims 1, 15, and 35 Under § 103

The dissertation “Dynamic Configuration Management of the Equipment in Distributed Communication Environments” by Sven van der Meer (Exhibit 8, “Meer96”) was published on October 6, 1996.

If certain aspects recited in claims 1, 15, and 35 of the '163 patent are not deemed to be disclosed, inherent, or obvious over Pfeifer96, then the inclusion of those aspects certainly would be obvious over Pfeifer96 in view of Meer96, under 35 U.S.C. § 103.

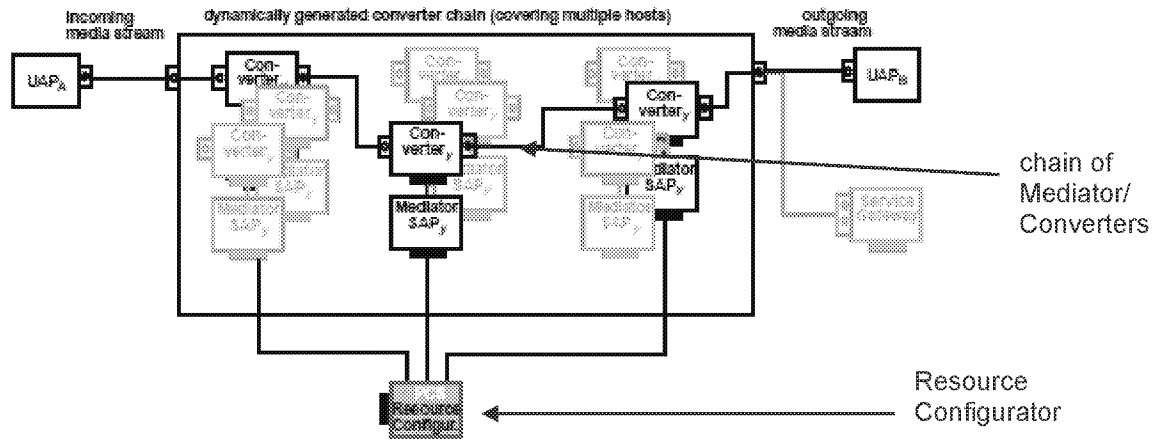
It was obvious to consider Pfeifer96 in view of Meer96 because both documents describe the “Intelligent Personal Communication Support System (iPCSS),” and Meer96 cites to Pfeifer96 for certain concepts. *See* Ex. 8 at 13, 15, 123 (citing Pfeifer96: “Generic Conversion of Communication Media for supporting Personal Mobility. To appear in the proceedings of the Third COST 237 Workshop . . . Nov 25-27, 1996”).

(a) Claim 1

Because Pfeifer96 and Meer96 are so similar in approach and detail, Meer96 reinforces and confirms the analysis presented in Section V.A.2 (Pfeifer96 103) in many ways. *See, e.g.*, Ex. 8 (Meer96) at 10 (“Dynamic Terminal Selection”), 14 (“Theory of Conversion”), 16 (“Generic Converter Model”), 69 (“Resource Configurator”).

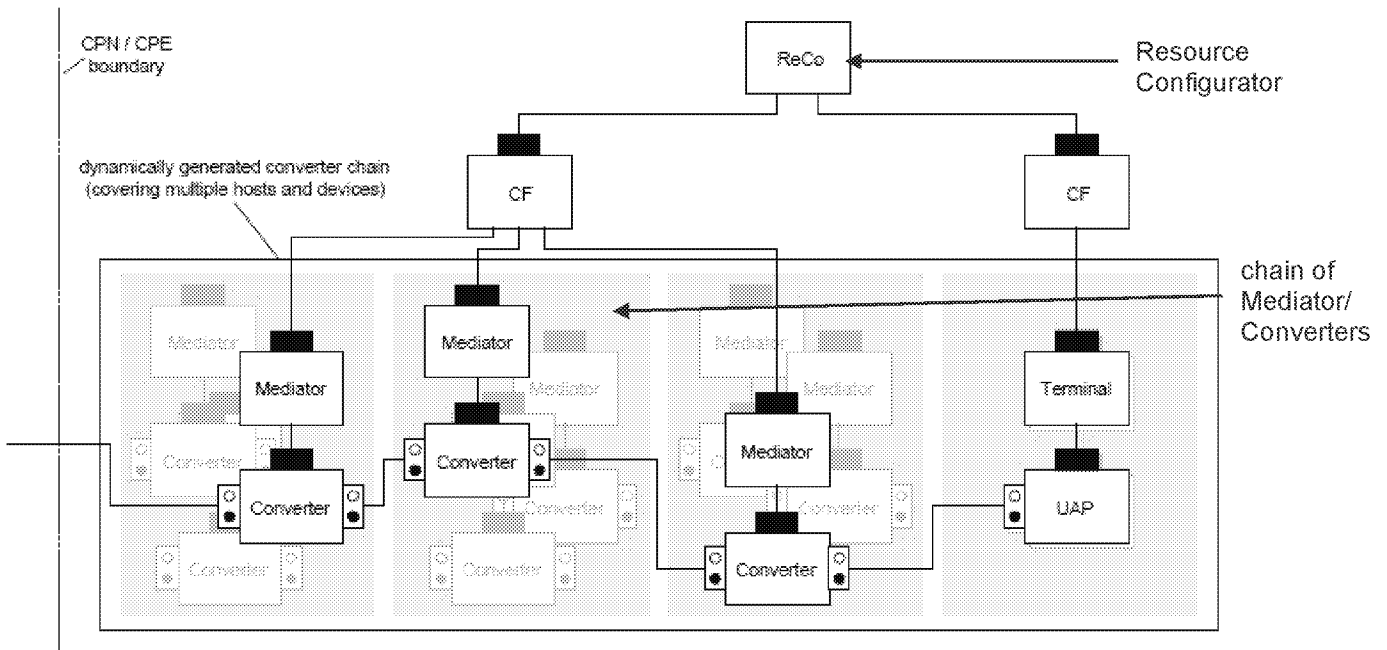
One particularly pertinent manner in which Meer96 presents significant additional information is regarding the “state information” element of claim 1 (“for each of a plurality of packets of the message in sequence, for each of a plurality of components in the identified non-predefined sequence, **retrieving** state information relating to performing the processing of the component with the previous packet of the message; **performing** the processing of the identified component with the packet and the retrieved state information; and **storing** state information relating to the processing of the component with the packet for use when processing the next packet of the message”).

Both Pfeifer96 and Meer96 teach that as a matter of internal architecture, each converter is managed by its own “Mediator”:



Ex. A02 (Pfeifer96) at 125 (Figure 12: “Converter chain, configured for a specific task,” showing “Resource Configurator” having orchestrated an arrangement wherein each “Converter” has its own “Mediator SAP”). See also, e.g., *id.* at 125 (“Each converter is subordinated to its specific MSAP,” which is “designed for the purpose of dynamic binding of converters”).

Meer96 echoes this general organization:



Ex. 8 (Meer96) at 73 (Figure 4-17: “Computational Modeling – Converter Chain,” showing “ReCo” [Resource Configurator] having created a “dynamically generated converter chain”

wherein each “Converter” has its own “Mediator”). *See also, e.g., id.* at 73-74 (each “Converter” in the diagram “represents the real [conversion] resource, either implemented in **software** or designed as hardware,” while each “Mediator” is “controlling software” which can “parameterize” its Converter and “configure [its] stream interfaces”) (emphasis added).

As suggested by the phrase “covering multiple hosts and devices” in Figure 4-17, Meer96 teaches that while not required, it is possible for particular “converter units developed as software” to reside on remote hosts. *Id.* at 73, 53-54. This flexibility of component positioning is possible because there exist “well defined and practical tested interworking of different network technologies.” *See id.* at 54. Specifically, Meer96 teaches that remotely positioned converters could be communicated with in at least three manners: (1) “via shared file-systems” as in “a UNIX operating system,” because “Network wide available files can be used to transmit data from one application to another, without any consideration about the host the software is running on”; (2) via “System V Inter-Process Communication (IPC),” which must “be taken into account as [a possible] transmission service”; and (3) via “seamless FTP and HTTP connections” which will be supported by “Future operating systems.” *Id.* at 53-55

Thus, using one of those three methods, the mediator for a remotely positioned converter can ensure that as data arrives via the converter chain for processing, the data can be routed to the remote converter and the remote converter can return its results. *See id.* at 53-54, 65-68.

One of ordinary skill would recognize that applying *any* of these three methods would require use of a *network transport protocol*, since the converter being communicated with is located across a network on a remote host. One of ordinary skill would find use of a network transport protocol from *the TCP/IP suite* to be the most obvious choice, both because it was the most popular protocol suite in the world at the time, and because it is integral to the “UNIX

operating system” cited by Meer⁹⁶ *See id.* at 53. There are two network transport protocols within TCP/IP—*i.e.*, TCP and UDP. Either would be an obvious choice, but *TCP* was especially obvious because it provides for reliable delivery, and because it is the *default* network transport protocol for “FTP” and “HTTP,” and is also commonly used for implementing “shared file-systems” in “UNIX” such as NFS (Network File System). *See id.*

Having made the obvious choice of TCP to communicate with a remote converter module using any of the three methods mentioned above, one of ordinary skill would recognize that an *ongoing TCP connection* from the mediator to the converter should be maintained, for performance reasons. Since a remote converter must be contacted to process every packet in an incoming stream, the additional overhead of opening and closing a new TCP connection to deliver every incoming packet would obviously be unattractive, and easily avoided by simply maintaining an ongoing TCP connection between the mediator and its remote converter.

And finally, one of ordinary skill would recognize that maintaining an ongoing TCP connection to a remote converter module would certainly require maintaining “state information relating to performing the processing of the component,” where “the component” can readily be seen as comprising the tightly integrated combination of a converter and its mediator. *See, e.g., id.* at 73-74.

For example, TCP includes an outgoing sequence number with each packet,¹⁴ so minimally, transmitting a packet from mediator to remote converter module would require, *e.g.*, **retrieving** the current sequence number for that ongoing connection, adding one to it, transmitting the packet, and then **storing** the new sequence number so it can be used for the next

¹⁴ *See, e.g.*, Ex. G25 (RFC 794: “Transmission Control Protocol”) (1981) (“RFC 793”) at 24 (section entitled “Sequence Numbers”: “A fundamental notion in the design is that every octet of data sent over a TCP connection has a sequence number.”). This reference is cited in this context solely to help explain the cited art. *See* MPEP § 2205.

packet. This sequence number is, *e.g.*, “state information relating to **performing** the process of the component” because an absolutely essential part of the processing performed by this component is delivering the data to the remote converter. For the same reason, this also reads on “performing the processing of the identified component with the packet and the retrieved state information,” since part of the essential **processing** of that component is transferring data from “the [incoming] “packet” to the remote converter, and this requires placing that data in a packet which is assigned a sequence number based on **retrieved** sequence number state information.

Thus, considering the most obvious implementation of any of the three methods disclosed by Meer96 for communicating with remote converters, these “state information” elements would be satisfied.

Claim 1 recites these “state information” elements in context of “for each of a plurality of packets of the message in sequence.” Because all of the packets in an incoming communication (“message”) would be processed by each component in the converter chain, *all* of the packets would be routed to any remote converter(s) in the chain.

Claim 1 also recites these “state information” elements in context of “for each of a plurality of components in the identified non-predefined sequence.” Meer96 places no restriction on the number of converters which could be positioned remotely, so it is obvious that any or all of the converters in a chain might be remotely positioned. *See id.* at 53-54, 65-68, 73-74.

(b) Claim 15

Again, because Pfeifer96 and Meer96 are so similar, Meer96 reinforces and confirms the analysis presented in Section V.A.2 (Pfeifer96 103) in many ways. *See, e.g.*, Meer96 at 10 (“Dynamic Terminal Selection”), 14 (“Theory of Conversion”), 16 (“Generic Converter Model”). 69 (“Resource Configurator”).

One particularly pertinent manner in which Meer96 presents significant additional information is regarding the “state information” element of claim 15 (“for each packet of each message, performing the processing of the identified non-predefined sequence of components of the message wherein state information generated by performing the processing of a component for a packet is available to the component when the component processes the next packet of the message”).

As discussed above under Claim 1, it was obvious that any or all of the components in a converter chain would include a remotely positioned converter, and it was obvious for any such component to maintain state information (*e.g.*, in form of an outgoing TCP sequence number) which would be updated and re-stored during the component’s processing of each packet of an incoming message. *See* Claim 1 above. Transferring data to the remotely positioned converter would be an essential part of the processing performed by such a component, and as explained above, this would entail updating a TCP sequence number (which is “state information generated by performing the processing of [the] component”), and would entail saving that current sequence number so that it may be “available to the component when the component processes the next packet of the message.”

(c) Claim 35

Again, because Pfeifer96 and Meer96 are so similar, Meer96 reinforces and confirms the analysis presented in Section V.A.2 (Pfeifer96 103) in many ways. *See, e.g.*, Meer96 at 10 (“Dynamic Terminal Selection”), 14 (“Theory of Conversion”), 16 (“Generic Converter Model”), 69 (“Resource Configurator”).

One particularly pertinent manner in which Meer96 presents significant additional information is regarding the “state information” element of claim 35 (“for each packet of the message, invoking the identified non-predefined sequence of components in sequence to perform

the processing of each component for the packet wherein each component saves message-specific state information so that that component can use the save message-specific state information when that component performs its processing on the next packet of the message”).

Pfeifer96 in view of Meer96 renders this element obvious. *See* Claim 15 (showing “for each packet of each message, performing the processing of the identified non-predefined sequence of components of the message wherein state information generated by performing the processing of a component for a packet is available to the component when the component processes the next packet of the message”) above.

8. Pfeifer96 in View of Meer96 and RFC 793 Renders Obvious Claims 1, 15, and 35 Under § 103

The specification RFC 793: “Transmission Control Protocol” (Ex. 9, “RFC 793”) by the Information Sciences Institute was published in September 1981. It was not considered during prosecution of the ‘163 patent.

If certain aspects recited in claims 1, 15, and 35 of the ‘163 patent are not deemed to be disclosed, inherent, or obvious over Pfeifer96 in view of Meer96, then the inclusion of those aspects certainly would be obvious over Pfeifer96 in view of Meer96 and RFC 793, under 35 U.S.C. § 103.

RFC 793 was cited above under MPEP § 2205 as confirming that certain background knowledge Pfeifer96 in view of Meer96 (immediately above). RFC 793 merely confirms that certain information regarding the stateful operation of TCP would have been part of the standard background knowledge of those of ordinary skill in the art.

It was also obvious to apply RFC 793 directly to Pfeifer96 and Meer96.

As RFC 793 is the original TCP specification, it would have been an obvious place to look for information regarding the specific operation of that protocol.

9. Pfeifer96 in View of Franz98 Renders Obvious Claims 1, 15, and 35 Under § 103

The dissertation “Job and Stream Control in Heterogeneous Hardware and Software Architectures” by Stefan Franz (Ex. 7, “Franz98”) was published on April 22, 1998, and it was not considered during prosecution of the ‘163 patent.

If certain aspects recited in claims 1, 15, and 35 of the ‘163 patent are not deemed to be disclosed, inherent, or obvious over Pfeifer96, then the inclusion of those aspects certainly would be obvious over Pfeifer96 in view of Franz98, under 35 U.S.C. § 103.

It was obvious to apply Franz98 to Pfeifer96 because the documents both describe the “Intelligent Personal Communication Support System (iPCSS),” with Franz98 presenting further detail regarding the control of jobs and streams in the iPCSS architecture. *See, e.g.*, Ex. 7 at v, 91-94.

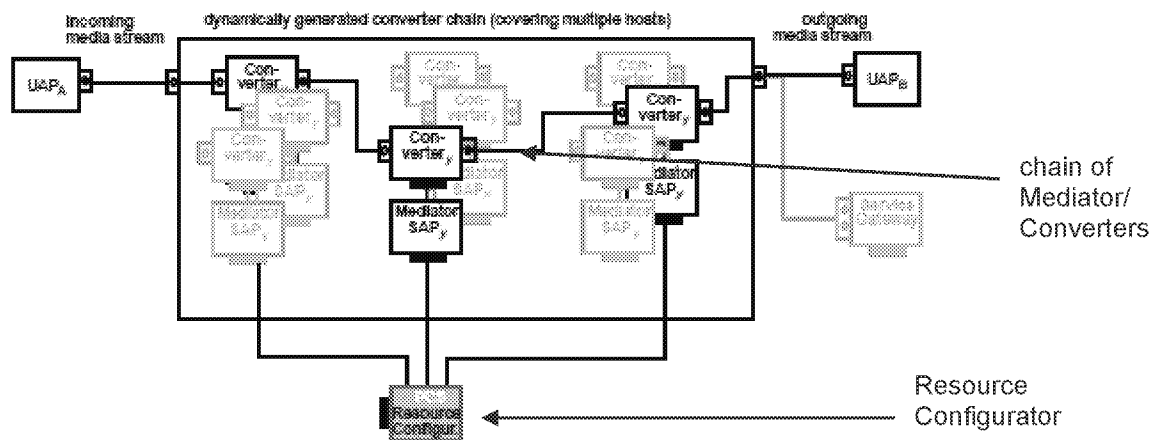
(a) Claim 1

Because Pfeifer96 and Franz98 are so similar in approach and detail, Franz98 reinforces and confirms the analysis presented in Section V.A.2 (Pfeifer96 103) in many ways. *See, e.g.*, Ex. 7 (Franz98) at 7-16 (including “Resource Configurator” and “Converter Framework”), 91-94 (including “Generic Converter Interface”).

One particularly pertinent manner in which Franz98 presents significant additional information is regarding the “state information” element of claim 1 (“for each of a plurality of packets of the message in sequence, for each of a plurality of components in the identified non-predefined sequence, **retrieving** state information relating to performing the processing of the component with the previous packet of the message; **performing** the processing of the identified component with the packet and the retrieved state information; and **storing** state information

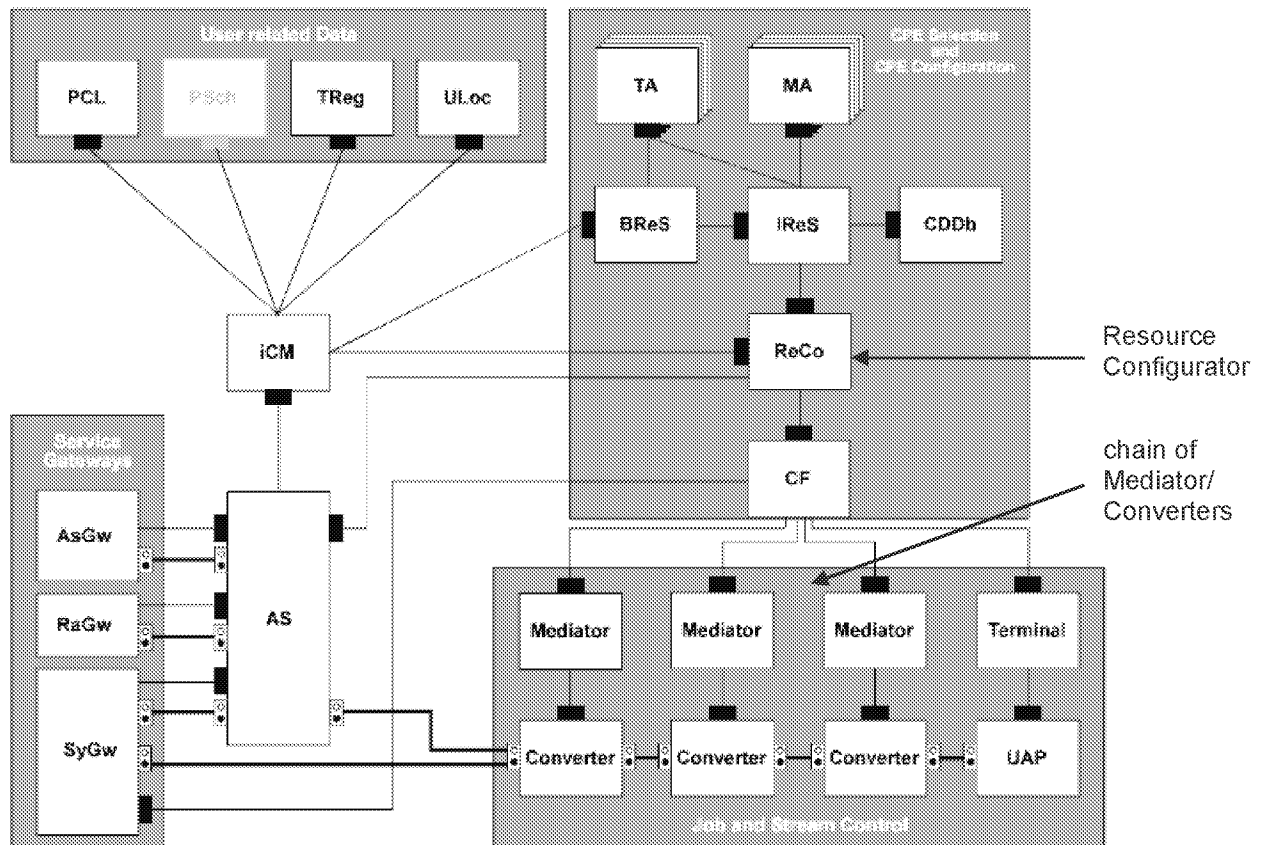
relating to the processing of the component with the packet for use when processing the next packet of the message”).

Both Pfeifer96 and Franz98 teach that as a matter of internal architecture, each converter is managed by its own “Mediator”:



Ex. A02 (Pfeifer96) at 125 (Figure 12: “Converter chain, configured for a specific task,” showing “Resource Configurator” having orchestrated an arrangement wherein each “Converter” has its own “Mediator SAP”). *See also, e.g., id.* at 124 (“Each converter is subordinated to its specific MSAP,” which is “designed for the purpose of dynamic binding of converters”).

Franz98 echoes this general organization:



Ex. 7 (Franz98) at 92 (Figure 6-1: “Location of Job and Stream Control within the context of the iPCSS”). *See also, e.g., id.* at 101-06.

Franz98 presents an elaborate and systematic analysis of the various software building blocks that would be needed for “Job Control and Stream Control” in the iPCSS. *See, e.g., id.* at 19 (“Types of Operating Systems” section), 50 (“Programs and Jobs” section”), 51 (“Processes” section), 55 (“Threads” section).

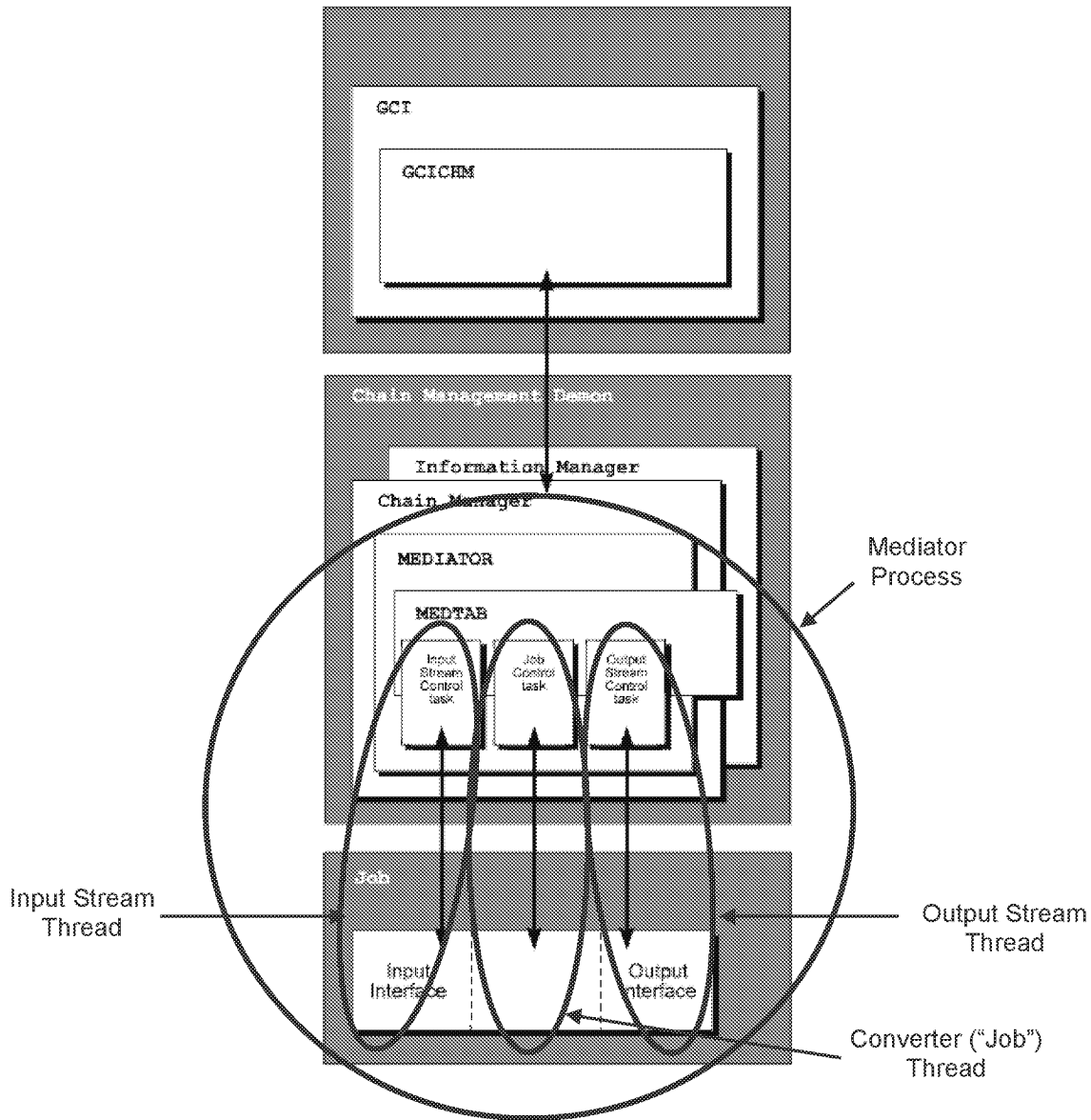
At the end of this lengthy analysis, Franz98 presents its conclusions about how the iPCSS should be structured, based on these building blocks. *See id.* at 91-112 (chapter entitled “Realisation”).

In this “Realisation” chapter, Franz98 recapitulates some iPCSS architectural concepts which would be familiar to readers of Pfeifer96, including that “one of the objectives of the

iPCSS is media conversion To provide media conversion more than one converter may be employed. The converters are enchainned and the result is named a converter chain.” *Id.* (Franz98) at 93.

Franz98 explains that because of the system’s focus, “the term job . . . and the term converter introduced by the iPCSS can be used synonymously” in its architectural analysis. *Id.*

Franz98 explains that “the control of a single converter” should be “encapsulated by an object named *Mediator*.” *Id.* at 93 (emphasis in original). When a Mediator object is created for use in a chain, “a new process” is created for that Mediator “containing three threads.” *Id.* at 101. Thus, each Mediator has its own process. As for the “three threads,” “one of these threads is associated to the Job Control [converter] where the other two implement the Stream Control” for the converter’s “input stream and . . . output stream.” *Id.* at 101.



Id. at 111 (Figure 6-9: “The final structure of the implementation”) (showing structure for a single mediator and its associated converter).

Franz98 explains how threads work internally. *See id.* at 50-66. In a “multitasking” system, it is “possible to load more than one program into memory, and these programs can “be executed concurrently.” *Id.* at 50. Multitasking can be implemented using processes and threads, and threads are “sometimes called lightweight processes.” *Id.* at 51, 55 (emphasis removed). Threads have “several states of execution” including “Running” (*e.g.*, while

processing incoming data) and “Blocked” (*e.g.*, while waiting for new data to arrive). *See id.* at 57, 50-55, 91-101 .

Essential information associated with a thread includes “the contents of the registers of the CPU” and “the current activity represented by the value of the program counter of the CPU.” *Id.* at 55. Those of ordinary skill understand that threads start and stop in the ordinary course of multitasking, and when a thread stops “The only thing to do is to **save the current activity and the used set of registers of the CPU**” (so they can be restored when execution of that thread resumes) *Id.* at 56 (emphasis added)

Thus, it would be clear to one of ordinary skill that when a converter thread is paused (*e.g.*, because it has finished processing a packet or for other reasons), “state information relating to the processing of the component” (including at least the current values of “the program counter of the CPU” and “the used set of registers of the CPU”) would be **stored**, in order that it can be **retrieved** and restored and used by the converter for **performing** its processing when the converter thread regains control (*e.g.*, upon receiving the next packet). *See id.* at 55-57, 91-101.

This state information (including at least “the program counter of the CPU” and the “used set of registers of the CPU”) is clearly “information relating to performing the processing of the component”: indeed, the program counter and registers would change in response to virtually every instruction performed in the course of the converter’s processing of a packet, so their state at the moment they were saved would clearly relate to that previous processing.

(b) *Claim 15*

Again, because Pfeifer96 and Franz98 are so similar, Franz98 reinforces and confirms the analysis presented in Section V.A.2 (Pfeifer96 103) in many ways. *See, e.g.*, Ex. 7 (Franz98) at 7-16 (including “Resource Configurator” and “Converter Framework”), 91-94 (including “Generic Converter Interface”).

One particularly pertinent manner in which Franz98 presents significant additional information is regarding the “state information” element of claim 15 (“for each packet of each message, performing the processing of the identified non-predefined sequence of components of the message wherein state information generated by performing the processing of a component for a packet is available to the component when the component processes the next packet of the message”).

As clear from the discussion above under Claim 1, it was obvious that the current values of the “program counter of the CPU” and the “the used set of registers of the CPU” are state information generated by performing the processing of a converter, and that because a converter has a thread, these values would be saved when the converter’s thread stops (*e.g.*, upon finishing its processing of a packet), and would be restored for use of the converter when the converter’s thread resumes (*e.g.*, upon receiving another packet to process).

(c) Claim 35

Again, because Pfeifer96 and Franz98 are so similar, Franz98 reinforces and confirms the analysis presented in Section V.A.2 (Pfeifer96 103) in many ways. *See, e.g.*, Ex. 7 (Franz98) at 7-16 (including “Resource Configurator” and “Converter Framework”), 91-94 (including “Generic Converter Interface”).

One particularly pertinent manner in which Franz98 presents significant additional information is regarding the “state information” element of claim 35 (“for each packet of the message, invoking the identified non-predefined sequence of components in sequence to perform the processing of each component for the packet wherein each component saves message-specific state information so that that component can use the save message-specific state information when that component performs its processing on the next packet of the message”).

Pfeifer96 in view of Franz98 renders this elements obvious. *See* Claim 15 (showing similar element) above.

10. Pfeifer96 in View of ISDN98, Nelson, Cox, Meer96, RFC 793, and Franz98 Renders Obvious Claims 1, 15, and 35 Under § 103

If certain aspects recited in claims 1, 15, and 35 of the '163 patent are not deemed to be disclosed, inherent, suggested, or obvious over Pfeifer96 alone or in combination with the various grounds of rejection presented above, then the inclusion of those aspects certainly would be obvious over Pfeifer96 in view of ISDN98, Nelson, Cox, Meer96, RFC 793, and Franz98 under 35 U.S.C. § 103, under Implicit's apparent claim constructions.

All of these references have already been combined with Pfeifer96 in corresponding sections above, and those sections should be consulted for the detailed manner of applying them to Pfeifer96. This section briefly summarizes that material and shows the collective combination of these references would be obvious as well.

Pfeifer96 teaches an "iPCSS" system wherein a converter chain is "dynamically generated" only after the first packet of a message is received. It also discloses and renders obvious that the converter components would maintain "state information" in the manners recited by claims 1, 15, and 35. It does so in several manners, including through use of ISDN connection converter components (which would maintain state information in order to execute the stateful ISDN protocol), and through use of components which perform compression or decompression. **ISDN98** confirms ISDN connections are stateful. **Nelson** confirms that obvious implementations of compression/decompression algorithms for use with Pfeifer96 would be stateful.

Cox teaches an “invocation-based metering” approach to software revenue collection which would be obvious to apply to Pfeifer96, and maintaining a cumulative invocation count would entail maintaining “state information.”

Meer97 explains that in the iPCSS system, a portion of every converter component could be located across a stateful network connection (e.g., a TCP connection), which would require maintaining “state information” for each. **RFC 793** confirms such a TCP connection would be stateful.

Franz98 explains that in the iPCSS system, every converter component would maintain state information across packets because of the operating system “threading” structure used for the converter component jobs.

In short, there is no aspect of claims 1, 15, and 35 which was not obvious over the prior art and combinations cited herein.

11. Pfeifer96 in View of Pfeifer97 and Alam Renders Obvious Claims 1, 15, and 35 Under § 103

U.S. Pat. No. 6,104,500 entitled “Networked Fax Routing Via Email” by Hassan Alam *et al.* (Exhibit 13, “Alam”) was filed on April 29, 1998, and it was not considered during prosecution of the ‘163 patent. It was obvious to apply Alam to Pfeifer96 and Pfeifer97 because both of these iPCSS documents describe scenarios in which an incoming fax is routed via a chain of converters to some other destination for consumption by the called party. *E.g.*, Ex. A02 (Pfeifer96) at 111, Ex. 12 (Pfeifer97) at 10.

As explained above, such incoming fax scenarios may clearly read on claims 1, 15, and 35. *E.g.*, Section V.A.1 (Pfeifer96 102) at Claim 1.

Alam confirms such routing would indeed be enabled, and supplies additional detail on how the called party of such an incoming fax could be obtained. Specifically, Alam teaches that

the fax image may be scanned, *e.g.*, “to locate name fields . . . based upon their nearness to and relationship with keywords. Keywords associated with the addressee’s name such as ‘To,’ ‘Recipient,’ ‘Attn’ or ‘Dear’ point to the addressee name.” Ex. 13 at 9:15-21. Once the destination party is determined, the iPCSS is clearly capable of determining that user’s location and routing the communication to a terminal in the user’s vicinity. *E.g.*, Ex. A02 at 119, 123-24.

12. Pfeifer96 in View of Pfeifer97 and Yun Renders Obvious Claims 1, 15, and 35 Under § 103

U.S. Pat. No. 5,298,576 entitled “Apparatus for Discriminating an Audio Signal as an Ordinary Vocal Sound or Musical Sound” by San-Lak Yun *et al.* (Exhibit 14, “Yun”) was filed on December 3, 1991 and issued on March 29, 1994. It was not considered during prosecution of the ‘163 patent. It was obvious to apply Alam to Pfeifer96 and Pfeifer97 because together the two iPCSS documents provide fuller detail on the iPCSS system, and because Pfeifer97 teaches a specific conversion between “audio” and “video” wherein the audio is displayed as “music notes on video.” Ex. 12 at 6.

By teaching an “apparatus for discriminating a received audio signal as vocal sound or musical sound,” Yun suggests how that specific conversion would be implemented and applied in practice. *See* Ex. 14 at Abstract. For example, example, incoming audio communications could be routed in one manner if they contain music (*e.g.*, to a screen for viewing “music notes on video”), and in another if they contain voice (*e.g.*, to a “speech recognition” component). *E.g.*, Ex. 12 at 6 (“display of music notes”; “speech recognition”). Such a conversion involving audio to video conversion would read on claims 1, 15, and 35. *See, e.g.*, Section V.A.1 (Pfeifer96 102) at Claim 1.

13. Pfeifer96 in View of Franz98, Meer96, Arbanowksi96, and Pfeifer97 Renders Obvious Claims 1, 15, and 35 Under § 103

If certain aspects recited in claims 1, 15, and 35 of the '163 patent are not deemed to be disclosed, inherent, suggested, or obvious over Pfeifer96 alone or in combination with the various grounds of rejection presented above, then the inclusion of those aspects certainly would be obvious over Pfeifer96 in view of Meer96, Franz98, Arbanowski96, and Pfeifer97, under 35 U.S.C. § 103, under Implicit's apparent claim constructions.

All of these references have already been combined with Decasper98 in corresponding sections above. Though each of these documents focuses on a different aspect of the iPCSS platform, collectively they provide a comprehensive picture of the system as a whole, its design, and possible uses.

14. Pfeifer96 in View of Arbanowski96, Pfeifer97, ISDN98, Nelson, Cox, Meer96, RFC 793, Franz98, Alam, and Yun Renders Obvious Claims 1, 15, and 35 Under § 103

If certain aspects recited in claims 1, 15, and 35 of the '163 patent are not deemed to be disclosed, inherent, suggested, or obvious over Pfeifer96 alone or in combination with the various grounds of rejection presented above, then the inclusion of those aspects certainly would be obvious over Pfeifer96 in view of Arbanowski96, Pfeifer97, ISDN98, Nelson, Cox, Meer96, RFC 793, Franz98, Alam, and Yun under 35 U.S.C. § 103, under Implicit's apparent claim constructions.

All of these references have already been combined with Pfeifer96 in corresponding sections above, and those sections should be consulted for the detailed manner of applying them to Pfeifer96. This section briefly summarizes that material and shows the collective combination of these references would be obvious as well.

Pfeifer96, Arbanowski96, Pfeifer97, Meer96, and Franz98 collectively provide a comprehensive picture of the iPCSS platform, including its design and possible uses.

Pfeifer96 teaches an “iPCSS” system wherein a converter chain is “dynamically generated” only after the first packet of a message is received. It also discloses and renders obvious that the converter components would maintain “state information” in the manners recited by claims 1, 15, and 35. It does so in several manners, including through use of ISDN connection converter components (which would maintain state information in order to execute the stateful ISDN protocol), and through use of components which perform compression or decompression. **ISDN98** confirms ISDN connections are stateful. **Nelson** confirms that obvious implementations of compression/decompression algorithms for use with Pfeifer96 would be stateful.

Cox teaches an “invocation-based metering” approach to software revenue collection which would be obvious to apply to Pfeifer96, and maintaining a cumulative invocation count would entail maintaining “state information.”

Meer97 explains that in the iPCSS system, a portion of every converter component could be located across a stateful network connection (e.g., a TCP connection), which would require maintaining “state information” for each. **RFC 793** confirms such a TCP connection would be stateful.

Franz98 explains that in the iPCSS system, every converter component would maintain state information across packets because of the operating system “threading” structure used for the converter component jobs.

Alam and **Yun** provide additional on how specific conversions might be implemented and applied in practice.

In short, there is no aspect of claims 1, 15, and 35 which was not obvious over the prior art and combinations cited herein.

B. Kerr (Exhibit 15)

U.S. Patent No. 6,243,667 entitled “Network Flow Switching and Flow Data Export” by Darren R. Kerr *et al.* (“Kerr”) was filed on May 28, 1996, and it was not considered during prosecution of the ‘163 patent.

1. Kerr Anticipates Claims 1, 15, and 35 Under § 102(e)

(a) Claim 1

i. “A method . . . for processing a message”

Claim 1 recites: “A method in a computer system for processing a message having a sequence of packets, the method comprising” Under Implicit’s apparent claim constructions, Kerr discloses this element.

As an initial matter, Kerr discloses “[a] method in a computer system.” For example, Kerr expressly states that that embodiments of its invention “may be implemented using a set of general purpose computers.” Ex. 15 at 2:30-32; *see also id.* at Figs. 1, 3 (illustrating data structures in computer network).

Claim 1 further recites the method is “for processing a message having a sequence of packets.” Kerr summarizes its invention in part as follows:

The invention provides a method and system for switching in networks responsive to message flow patterns. *A message “flow” is defined to comprise a set of packets* to be transmitted between a particular source and a particular destination. When routers in a network identify a new message flow, they determine the proper *processing for packets* in that message flow

Id. at 1:48-55 (emphasis added).

ii. “a plurality of components”

Claim 1 further recites: “providing a plurality of components, each component being a software routine for converting data with an input format into data with an output format.”

Under Implicit’s apparent claim constructions, Kerr discloses this element.

Kerr discloses a “plurality of components” for processing messages. For example, claim 1 of Kerr describes using a “plurality of devices” to apply “policy treatments” to a “plurality of messages,” where policy treatments are used to perform “access control,” “security,” “queuing,” “accounting,” “traffic profiling,” etc. *Id.* at 10:27-40. Processing components can include “treatment with regard to switching,” “access control,” and “encryption.” *Id.* at 4:20-34. “[S]pecial processing” can include “authentication” techniques “useful for implementing security ‘firewalls.’” *Id.* at 35-46. Kerr further discloses that a “rewrite function” may be invoked “to alter the header for the packet.” *Id.* at 4:55-62.

These components can be used for “converting data with an input format into data with an output format,” under Implicit’s apparent claim constructions, for example (as described above), the “encryption” and “rewrite” components to “alter” data to be processed. *Id.* at 4:30-31, 4:55-62.

The processing components of Kerr comprise “software routine” embodiments, as Kerr states that the processing instrumentality “may include specific hardware constructed *or programmed* performing the process steps described herein” or “a general purpose processor *operating under program control.*” *Id.* at 2:51-55; *see also id.* at Figs. 3-4 (illustrating software data structures).

iii. “dynamically identifying a non-predefined sequence”

Claim 1 further recites: “for the first packet of the message, dynamically identifying a non-predefined sequence of components for processing the packets of the message such that the output format of the components of the non-predefined sequence match the input format of the

next component in the non-predefined sequence, wherein dynamically identifying includes selecting individual components to create the non-predefined sequence of components after the first packet is received.” Under Implicit’s apparent claim constructions, Kerr discloses this element.

Implicit has characterized the “dynamically identifying” element as encompassing the ability of a network “administrator” to modify or create “Policy Files to change how traffic is managed at runtime.” Ex. 37-D [Implicit Technical Tutorial] at 35; *see generally id.* at 26-42. For example, Implicit has applied this claim construction to the example of a “system administrator” who can “dynamically” implement changing policies to block or permit access to YouTube for certain times or users:

The beauty – and object – of the Implicit system lay in its flexibility. Since a stateful path was not identified and instantiated until *post-first packet*, the system could be changed, *dynamically* on the fly. New components could be added, new rules or policies developed, all as new needs arose. For example, a system administrator could decide how to process particular types of traffic (no You Tube between noon and one) and then *change the rules – or policies* – the next minute or the next day (only CEO gets You Tube).

Ex. 37-A [Implicit Opening Claim Construction Brief] at 5.

Kerr discloses “dynamically identifying” under Implicit’s apparent claim construction. Kerr explains that a “message flow may be identified responsive to relative network congestion or administrative policies.” Ex. 15 at 3:28-34. One specific example Kerr provides of “enforcing administrative policies” is to “monitor . . . access using the HTTP protocol to world wide web pages at particular sites.” *Id.* at 5:33-40. Policies may be applied and information collected about access to “web page[s] in response to date and time of access,” including parameters regarding “HTTP access” during “particular dates or times,” particularly “accesses which occur outside normal working hours.” *Id.* at 9:34-60. Another example in Kerr

is to “monitor usage information regarding relative use of network resources” in real time for packet prioritization purposes. *Id.* at 5:41-49.

Kerr also makes clear that administrators can make other rule-based or policy changes during *runtime*, which falls within the scope of “dynamically identifying a non-predefined sequence of components” under Implicit’s apparent claim construction. *See* Section IV.C. For example, Kerr evaluates “changed conditions” that may occur during the lifetime of a flow (*i.e.*, after the first packet of that flow), such as “*changes in access control lists* or other changes which might affect the proper treatment of packets 150 in the message flow 160.” Ex. 15 at 6:13-18. If a runtime change in an access control list so mandates, a flow will be “expired” in the flow cache. *Id.*

The components disclosed in Kerr are used in a manner “such that the output format of the components . . . match the input format of the next component,” under Implicit’s apparent claim construction. *See* Section IV.C. The Kerr system is properly capable of handling traffic in “ethernet,” “IP,” “TCP,” and “UDP” formats (Ex. 15 at 2:48, 3:1, Fig. 4 (“IP address cache”)), in addition to application-layer protocols such as “HTTP protocol or the FTP protocol” (*id.* at 9:54-55). As explained above, packets may be processed by multiple components in the course of being successfully “transmitted between particular pairs of transport service access points.” *Id.* at 2:57-58. Because packets compatibly move from component to component, this element is satisfied under Implicit’s apparent claim construction.

The “dynamically identifying” as disclosed in Kerr (under Implicit’s apparent claim construction) also “includes selecting individual components to create the non-predefined sequence of components after the first packet is received.” As described above, the components disclosed in Kerr are “select[ed]” as “individual components” (encryption, rewrite, etc.)

associated with a particular flow entry. Figure 2A of Kerr, for example, shows the step “BUILD NEW ENTRY,” which is executed *after* the step of “RECEIVE PACKET” and occurs only if the packet is identified as “NEW” (*i.e.*, it is the first packet in a new flow). *Id.* at Fig. 2A. Implicit has indicated that instantiating in memory constitutes “creating” for purposes of the patent. *See* Section IV.C. Kerr discloses a “flow cache” that “comprises a memory,” in which message flows are “cached” when the system “identif[ies] a new message flow.” Ex. 15 at 1:52-54, 6:32.

iv. “storing an indication of . . . the identified components”

Claim 1 further recites “storing an indication of each of the identified components so that the non-predefined sequence does not need to be re-identified for subsequent packets of the message.” Under Implicit’s apparent claim constructions, Kerr discloses this element.

As explained above, after receiving the first packet of a new flow, Kerr builds a new flow entry that is cached in memory, which constitutes “storing” under Implicit’s apparent claim construction. Kerr also explains that building and caching a flow entry upon receiving the first new packet in a flow is specifically performed so that information “does not need to be re-identified for subsequent packets of the message,” as that term is apparently construed by Implicit. Kerr explains that, for the sake of efficiency:

information about message flow patterns is used to identify packets for which processing has *already been determined*, and therefore to process those packets *without having to re-determine* the same processing

Thus, in a preferred embodiment, the routing device 140 *does not separately determine*, for each packet 150 in the message flow 160, the information stored in the entry in the flow cache. Rather, when routing a packet 150 in the message flow 160, the routing device 140 *reads the information from the entry in the flow cache* and treats the packet 150 according to the information in the entry in the flow cache.

Ex. 15 at 1:33-36, 4:64-5:4 (emphasis added).

In other words, when the first packet of a flow arrives, Kerr goes through the somewhat expensive and elaborate process of determining how *all* the packets of that flow should be treated: *e.g.*, whether they should be encrypted, whether they should be modified or partially rewritten, and where they should be routed next. *Id.* at 1:33-35, 4:13-60. It then records all this information about the proper processing for a flow by “build[ing] a new entry in the flow cache” for the flow, so the proper processing does not have to be wastefully and redundantly determined again for subsequent packets of the flow. *Id.* at 4:12-13.

v. “state information”

Claim 1 further recites “for each of a plurality of packets of the message in sequence, for each of a plurality of components in the identified non-predefined sequence, **retrieving** state information relating to performing the processing of the component with the previous packet of the message; **performing** the processing of the identified component with the packet and the retrieved state information; and **storing** state information relating to the processing of the component with the packet for use when processing the next packet of the message.” Under Implicit’s apparent claim constructions, Kerr discloses this “state information” element.

Implicit has taken a broad view of the “state information” limitations, arguing that they cover the retrieval, use, and storage of the identified sequence of components (*e.g.*, a flow record) after the first packet is received. *See* Section IV.C. As demonstrated above (for the “storing an indication” element), Kerr retrieves, uses, and stores flow records in this manner to facilitate processing of packets in the same message after the first packet is received and a flow entry built.

Kerr also discloses the retrieval, use, and storage of state information on a component-by-component basis. For example, in one embodiment of Kerr, there are components for access control, encryption, “special treatment,” accounting, rewrite, among others. Ex. 15 at 5:5-25.

The processing by these components is “all responsive to information in the entry in the flow cache.” *Id.* at 5:9-10. As a specific example, an accounting component can maintain state information, such as “time stamp” data, “a cumulative count for the number of packets,” and “a cumulative count for the number of bytes.” *Id.* at 6:58-63. Kerr later uses timing information to identify expired or otherwise invalid flows (among other reasons). *Id.* at 5:52 – 6:19. As another example, Kerr can retrieve the latest “usage information regarding relative use of network resources” in order to appropriately prioritize traffic using the relevant component. *Id.* at 5:41-49. These would also satisfy the “state information” limitations under Implicit’s apparent claim construction.

(b) Claim 15

i. “demultiplexing packets of messages”

Claim 15 recites: “A method in a computer system demultiplexing packets of messages.” Under Implicit’s apparent claim constructions, Kerr discloses this element. *See* Claim 1(i) (showing “A method in a computer system for processing a message having a sequence of packets”) above. Under Implicit’s apparent claim constructions, “demultiplexing” a packet is satisfied by routing a packet to the correct sequence of components for processing it—and Kerr performs this function. *See* Section IV.C. and, *e.g.*, Claim 1(iii) (showing “dynamically identifying a non-predefined sequence of components for processing the packets of the message”) above.

ii. “dynamically identifying a non-predefined sequence”

Claim 15 further recites “dynamically identifying a non-predefined sequence of components for processing each message based on the first packet of the message so that subsequent packets of the message can be processed without re-identifying the components.” Under Implicit’s apparent claim constructions, Kerr discloses this element. *See* Claim 1(iii)

(showing “for the first packet of the message, dynamically identifying a non-predefined sequence of components for processing the packets of the message”) and Claim 1(iv) (showing “storing an indication of each of the identified components so that the non-predefined sequence does not need to be re-identified for subsequent packets of the message”) above.

iii. “different . . . sequences of components can be identified”

Claim 15 further recites “different non-predefined sequences of components can be identified for different messages.” Under Implicit’s apparent claim constructions, Kerr discloses this element. Rather than applying a single predefined sequence to all flows, Kerr “determines proper treatment of packets 150 in the message flow” only when it “build a new entry in the flow cache” for that specific flow. Ex. 15 at 4:12-18. This includes, *e.g.*, “determin[ing] encryption treatment for packets 150 in the message flow . . . and any special treatment for packets 150 in the message flow.” *Id.* at 4:31-34..

iv. “each component being a software routine”

Claim 15 further recites “each component being a software routine.” Under Implicit’s apparent claim constructions, Kerr discloses this element. *See* Claim 1(ii) (showing “each component being a software routine”) above.

v. “selecting individual components”

Claim 15 further recites “dynamically identifying includes selecting individual components to create the non-predefined sequence of components.” Under Implicit’s apparent claim constructions, Kerr discloses this element. *See* Claim 1(iii) (showing “dynamically identifying includes selecting individual components to create the non-predefined sequence of components”) above.

vi. “state information”

Claim 15 finally recites “for each packet of each message, performing the processing of the identified non-predefined sequence of components of the message wherein state information generated by performing the processing of a component for a packet is available to the component when the component processes the next packet of the message.” Under Implicit’s apparent claim constructions, Kerr discloses this element. *See* Claim 1(v) (showing similar “state information” element) above.

(c) *Claim 35*

i. *“instructions for demultiplexing packets of messages”*

Claim 35 recites: “A computer-readable medium containing instructions for demultiplexing packets of messages.” Under Implicit’s apparent claim constructions, Kerr discloses this element. Kerr teaches its invention may be performed by “a general purpose processor operating under program control,” and it would be understood by those of ordinary skill that such a controlling program would be loaded from a “computer-readable medium,” such as a hard disk in the routing device. Ex. 15 at 2:53-54. *See also* Claim 15(i) showing (“demultiplexing packets of messages”) above.

ii. *“dynamically identifying a . . . non-predefined sequence”*

Claim 35 recites: “dynamically identifying a message-specific non-predefined sequence of components for processing the packets of each message upon receiving the first packet of the message.” Under Implicit’s apparent claim constructions, Kerr discloses this element. *See* Claim 1(iii) (showing “for the first packet of the message, dynamically identifying a non-predefined sequence of components for processing the packets of the message”) above.

iii. *“subsequent packets . . . can use the . . . sequence”*

Claim 35 further recites that “subsequent packets of the message can use the message-specific non-predefined sequence identified when the first packet was received.” Under

Implicit's apparent claim constructions, Kerr discloses this element. *See* Claim 1(iv) (showing “storing an indication of each of the identified components so that the non-predefined sequence does not need to be re-identified for subsequent packets of the message”) above.

iv. “selecting individual components”

Claim 35 further recites that “dynamically identifying includes selecting individual components to create the message-specific non-predefined sequence of components.” Under Implicit's apparent claim constructions, Kerr discloses this element. *See* Claim 1(iii) (showing “dynamically identifying includes selecting individual components to create the non-predefined sequence of components after the first packet is received”) above.

v. “state information”

Claim 35 further recites “for each packet of the message, invoking the identified non-predefined sequence of components in sequence to perform the processing of each component for the packet wherein each component saves message-specific state information so that that component can use the save message-specific state information when that component performs its processing on the next packet of the message.” Under Implicit's apparent claim constructions, Kerr discloses this element. *See* Claim 1(v) (showing similar “state information” element) above.

2. Kerr Renders Obvious Claims 1, 15, and 35 Under § 103

If certain aspects recited in claims 1, 15, and 35 of the '163 patent are not deemed to be disclosed or inherent over Kerr, then the inclusion of those aspects certainly would be obvious over Kerr, under 35 U.S.C. § 103.

(a) Claim 1

i. “A method . . . for processing a message”

Claim 1 recites: “A method in a computer system for processing a message having a sequence of packets, the method comprising” Under Implicit’s apparent claim constructions, Kerr renders obvious this element.

Because Kerr teaches its invention may be performed by “a general purpose processor operating under program control,” it was obvious to perform the method in a computer system. Ex. 15 at 2:53-54.

ii. “a plurality of components”

Claim 1 further recites: “providing a plurality of components, each component being a software routine for converting data with an input format into data with an output format.” Under Implicit’s apparent claim constructions, Kerr renders obvious this element.

Kerr discloses a plurality of distinct operations which may be performed on the packets of a flow. For example, claim 1 of Kerr describes using a “plurality of devices” to apply “policy treatments” to a “plurality of messages,” where policy treatments are used to perform “access control,” “security,” “queuing,” “accounting,” “traffic profiling,” etc. *Id.* at 10:27-40. Processing components can include “treatment with regard to switching,” “access control,” and “encryption.” *Id.* at 4:20-34. “[S]pecial processing” can include “authentication” techniques “useful for implementing security ‘firewalls.’” *Id.* at 35-46. Kerr further discloses that a “rewrite function” may be invoked “to alter the header for the packet.” *Id.* at 4:55-62.

Because these are distinct operations, it was obvious to implement them as distinct software routines in accordance with ordinary programming practice. *Id.* at 2:53-54 (“general purpose processor . . . under program control”). Indeed, Kerr expressly mentions its rewrite operation would be implemented as such. *Id.* at 4:56-57 (“the flow cache includes a pointer to a rewrite *function*”) (emphasis added).

These components can be used for “converting data with an input format into data with an output format,” under Implicit’s apparent claim constructions, for example (as described above), the “encryption” and “rewrite” components to “alter” data to be processed. *Id.* at 4:30-31, 4:55-62.

iii. “*dynamically identifying a non-predefined sequence*”

Claim 1 further recites: “for the first packet of the message, dynamically identifying a non-predefined sequence of components for processing the packets of the message such that the output format of the components of the non-predefined sequence match the input format of the next component in the non-predefined sequence, wherein dynamically identifying includes selecting individual components to create the non-predefined sequence of components after the first packet is received.” Under Implicit’s apparent claim constructions, Kerr renders obvious this element.

Implicit has characterized the “dynamically identifying” element as encompassing the ability of a network “administrator” to modify or create “Policy Files to change how traffic is managed at runtime.” Ex. 37-D [Implicit Technical Tutorial] at 35; *see generally id.* at 26-42. For example, Implicit has applied this claim construction to the example of a “system administrator” who can “dynamically” implement changing policies to block or permit access to YouTube for certain times or users:

The beauty – and object – of the Implicit system lay in its flexibility. Since a stateful path was not identified and instantiated until *post-first packet*, the system could be changed, *dynamically* on the fly. New components could be added, new rules or policies developed, all as new needs arose. For example, a system administrator could decide how to process particular types of traffic (no You Tube between noon and one) and then *change the rules – or policies* – the next minute or the next day (only CEO gets You Tube).

Ex. 37-A [Implicit Opening Claim Construction Brief] at 5.

Kerr discloses “dynamically identifying” under Implicit’s apparent claim construction. Kerr explains that a “message flow may be identified responsive to . . . relative network congestion or administrative policies.” Ex. 15 at 3:28-34. One specific example Kerr provides of “enforcing administrative policies” is to “monitor . . . access using the HTTP protocol to world wide web pages at particular sites.” *Id.* at 5:33-40. Policies may be applied and information collected about access to “web page[s] in response to date and time of access,” including parameters regarding “HTTP access” during “particular dates or times,” particularly “accesses which occur outside normal working hours.” *Id.* at 9:34-60. Another example in Kerr is to “monitor usage information regarding relative use of network resources” in real time for packet prioritization purposes. *Id.* at 5:41-49.

Kerr also makes clear that administrators can make other rule-based or policy changes during *runtime*, which falls within the scope of “dynamically identifying a non-predefined sequence of components” under Implicit’s apparent claim construction. *See* Section IV-C. For example, Kerr evaluates “changed conditions” that may occur during the lifetime of a flow (*i.e.*, after the first packet of that flow), such as “*changes in access control lists* or other changes which might affect the proper treatment of packets 150 in the message flow 160.” Ex. 15 at 6:13-18. If a runtime change in an access control list so mandates, a flow will be “expired” in the flow cache. *Id.*; *see also, e.g., id.* at 8:42-44 (after being “initially configured,” routing device parameters “may be altered by an operator”).

Regarding the limitation “such that the output format of the components . . . match the input format of the next component,” it was well-known to those of ordinary skill in the art that certain operations on a packet must be performed in a certain order: *e.g.*, if a packet is first converted into an encrypted format by a first component, a subsequent component would be

unable to, *e.g.*, rewrite its headers (because it was expecting to receive the packet in an unencrypted format). *See id.* at 4:31-32 (“encryption treatment for packets . . . in the message flow”), 4:57-58 (“rewrite function for . . . a header for the packet”). Thus, it was certainly at least obvious for one of ordinary skill in the art to arrange the sequence of components in a compatible manner, such that the output format of one matches the input format of the next—rather than arranging them in an incompatible manner whereby various component(s) would be unable to perform their function(s).

Regarding the limitation “wherein dynamically identifying includes selecting individual components,” as explained above it was obvious to implement each distinct (and individual) operation performed on a flow as a distinct (and individual) software routine. *See* Claim 1(ii) (showing “each component being a software routine”) above.

The “dynamically identifying” as disclosed in Kerr (under Implicit’s apparent claim construction) also “includes selecting individual components to create the non-predefined sequence of components after the first packet is received.” As described above, the components disclosed in Kerr are “select[ed]” as “individual components” (encryption, rewrite, etc.) associated with a particular flow entry. Figure 2A of Kerr, for example, shows the step “BUILD NEW ENTRY,” which is executed *after* the step of “RECEIVE PACKET” and occurs only if the packet is identified as “NEW” (*i.e.*, it is the first packet in a new flow). *Id.* at Fig. 2A. Implicit has indicated that instantiating in memory constitutes “creating” for purposes of the patent. *See* Section IV-C. Kerr discloses a “flow cache” that “comprises a memory,” in which message flows are “cached” when the system “identif[ies] a new message flow.” Ex. 15 at 1:52-54, 6:32.

iv. “storing an indication of . . . the identified components”

Claim 1 further recites “storing an indication of each of the identified components so that the non-predefined sequence does not need to be re-identified for subsequent packets of the message.” Under Implicit’s apparent claim constructions, Kerr renders obvious this element.

Kerr explains that when the first packet of a new flow arrives, the system “builds a new entry in the flow cache” which “determines proper treatment of packets 150 in the message flow and *enters information regarding such treatment in a data structure* pointed to by the new entry in the flow cache” *Id.* at 4:12-16 (emphasis added).

As explained above, it was obvious to implement the various distinct operations which may comprise this “proper treatment” as distinct software routines. *See* Claim 1(ii) above.

Such “proper treatment” would, then, consist of the selected sequence of these software routines, so it was obvious (and necessary) that the “information regarding such treatment” would comprise an indication of the selected routines. *Id.* at 4:12-16. Indeed, Kerr expressly mentions doing so for the rewrite function, and there is no indication in Kerr that other routines would or should be treated differently. *Id.* at 4:56-57 (“the flow cache includes a pointer to a rewrite *function*”) (emphasis added).

v. “state information”

Claim 1 further recites “for each of a plurality of packets of the message in sequence, for each of a plurality of components in the identified non-predefined sequence, **retrieving** state information relating to performing the processing of the component with the previous packet of the message; **performing** the processing of the identified component with the packet and the retrieved state information; and **storing** state information relating to the processing of the component with the packet for use when processing the next packet of the message.” Under Implicit’s apparent claim constructions, Kerr renders obvious this “state information” element.

Implicit has taken a broad view of the “state information” limitations, arguing that they cover the retrieval, use, and storage of the identified sequence of components (*e.g.*, a flow record) after the first packet is received. *See* Section IV.C. As demonstrated above (for the “storing an indication” element), Kerr retrieves, uses, and stores flow records in this manner to facilitate processing of packets in the same message after the first packet is received and a flow entry built.

Kerr also renders obvious the retrieval, use, and storage of state information on a component-by-component basis. For example, in one embodiment of Kerr, there are components for access control, encryption, “special treatment,” accounting, rewrite, among others. Ex. 15 at 5:5-25. The processing by these components is “all responsive to information in the entry in the flow cache.” *Id.* at 5:9-10. As a specific example, an accounting component can maintain state information, such as “time stamp” data, “a cumulative count for the number of packets,” and “a cumulative count for the number of bytes.” *Id.* at 6:58-63. Kerr later uses timing information to identify expired or otherwise invalid flows (among other reasons). *Id.* at 5:52 – 6:19. As another example, Kerr can retrieve the latest “usage information regarding relative use of network resources” in order to appropriately prioritize traffic using the relevant component. *Id.* at 5:41-49.

As another example, an obvious implementation of the “encryption” component would read on this “state information” element. Kerr supports “IP,” the Internet Protocol, and one of ordinary skill would be aware that an encryption algorithm which “MUST” be supported by the security architecture for the Internet Protocol is specified in RFC 1829.¹⁵ *Id.* at 3:5 (“IP (internet

¹⁵ *See* Ex. 26 (RFC 1825: “Security Architecture for the Internet Protocol”) (1995) at 10 (the encryption operation “MUST support the use of the Data Encryption Standard (DES) in Cipher-Block Chaining (CBC) Mode”), 21 (citing “RFC 1829”: “The ESP DES-CBC

protocol”). RFC 1829 explains that in order to apply its encryption algorithm, “an Initialization Vector (IV) that is eight octets in length” must be placed in “[e]ach datagram” to be encrypted (*i.e.*, in each packet). *See* Ex. 27 (RFC 1829: “The ESP DES-CBC Transform” by P. Karn et al.) (1995) at 1. RFC 1829 further explains that while the “method for selection of IV values is implementation dependent,” a “common acceptable technique is simply a counter, beginning with a random chosen value.” *Id.* One of ordinary skill would therefore find it obvious to apply this technique to implement the encryption component of Kerr. Doing so would clearly entail, for each packet: *e.g.*, retrieving the previous counter value, applying it the packet it, incrementing the counter value, and storing it for use when encrypting the next packet. Under Implicit’s apparent claim constructions, this obvious implementation would read on this “state information” claim element.

(b) *Claim 15*

i. *“demultiplexing packets of messages”*

Claim 15 recites: “A method in a computer system demultiplexing packets of messages.” Under Implicit’s apparent claim constructions, Kerr discloses this element. *See* Section V.B.1 (Kerr 102) at Claim 15(i) (showing same element) above.

ii. *“dynamically identifying a non-predefined sequence”*

Claim 15 further recites “dynamically identifying a non-predefined sequence of components for processing each message based on the first packet of the message so that subsequent packets of the message can be processed without re-identifying the components.” Under Implicit’s apparent claim constructions, Kerr discloses this element. *See* Section V.B.1 (Kerr 102) at Claim 15(ii) (showing same element) above.

Transform”). RFC 1825 and RFC 1829 are cited in this section solely to help explain Kerr. *See* MPEP § 2205.

iii. *“different . . . sequences of components can be identified”*

Claim 15 further recites “different non-predefined sequences of components can be identified for different messages.” Under Implicit’s apparent claim constructions, Kerr discloses this element. *See* Section V.B.1 (Kerr 102) at Claim 15(iii) (showing same element) above.

iv. *“each component being a software routine”*

Claim 15 further recites “each component being a software routine.” Under Implicit’s apparent claim constructions, Kerr renders obvious this element. *See* Claim 1(ii) (showing same element) above.

v. *“selecting individual components”*

Claim 15 further recites “dynamically identifying includes selecting individual components to create the non-predefined sequence of components.” Under Implicit’s apparent claim constructions, Kerr renders obvious this element. *See* Claim 1(iii) (showing same element) above.

vi. *“state information”*

Claim 15 finally recites “for each packet of each message, performing the processing of the identified non-predefined sequence of components of the message wherein state information generated by performing the processing of a component for a packet is available to the component when the component processes the next packet of the message.” Under Implicit’s apparent claim constructions, Kerr renders obvious this element. *See* Claim 1(v) (showing similar “state information” element) above.

(c) *Claim 35*

i. *“instructions for demultiplexing packets of messages”*

Claim 35 recites: “A computer-readable medium containing instructions for demultiplexing packets of messages.” Under Implicit’s apparent claim constructions, Kerr

renders obvious this element. Kerr teaches that its invention may be performed by “a general purpose processor operating under program control,” and it was obvious to load such a controlling program from a “computer-readable medium,” such as a hard disk in the routing device. Ex. 15 at 2:53-54. *See also* Section V.B.1 (Kerr 102) at Claim 15(i) (showing “demultiplexing packets of messages”) above.

ii. “dynamically identifying a . . . non-predefined sequence”

Claim 35 recites: “dynamically identifying a message-specific non-predefined sequence of components for processing the packets of each message upon receiving the first packet of the message.” Under Implicit’s apparent claim constructions, Kerr renders obvious this element. *See* Claim 1(iii) (showing “for the first packet of the message, dynamically identifying a non-predefined sequence of components for processing the packets of the message”) above.

iii. “subsequent packets . . . can use the . . . sequence”

Claim 35 further recites that “subsequent packets of the message can use the message-specific non-predefined sequence identified when the first packet was received.” Under Implicit’s apparent claim constructions, Kerr renders obvious this element. *See* Claim 1(iv) (showing “storing an indication of each of the identified components so that the non-predefined sequence does not need to be re-identified for subsequent packets of the message”) above.

iv. “selecting individual components”

Claim 35 further recites that “dynamically identifying includes selecting individual components to create the message-specific non-predefined sequence of components.” Under Implicit’s apparent claim constructions, Kerr renders obvious this element. *See* Claim 1(iii) (showing same element) above.

v. “state information”

Claim 35 further recites “for each packet of the message, invoking the identified non-predefined sequence of components in sequence to perform the processing of each component for the packet wherein each component saves message-specific state information so that that component can use the save message-specific state information when that component performs its processing on the next packet of the message.” Under Implicit’s apparent claim constructions, Kerr renders obvious this element. *See* Claim 1(v) (showing similar “state information” element) above.

3. Kerr in View of NetFlow Renders Obvious Claims 1, 15, and 35 Under § 103

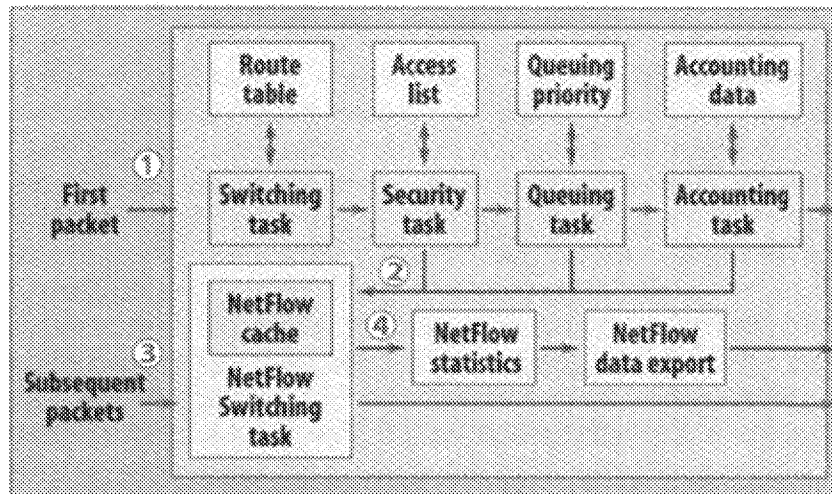
The article “Cisco NetFlow Switching speeds traffic routing” (Ex. 16, “NetFlow”) by Stephen Lawson was published on July 7, 1997 in the publication InfoWorld. NetFlow was not considered during the prosecution of the ‘163 patent.

If certain aspects recited in claims 1, 15, and 35 of the ‘163 patent are not deemed to be disclosed, inherent, or obvious over Mosberger, then the inclusion of those aspects certainly would be obvious over Kerr in view of NetFlow, under 35 U.S.C. § 103.

It was obvious to supplement the teachings of Kerr with NetFlow because Kerr is a Cisco patent, and NetFlow is an article in a trade publication illustrating how the architecture of Kerr manifested itself in an actual Cisco product feature (named “NetFlow”) that was available on the market within the same time period.

The following illustrative figure appears in NetFlow as a description of the technology:

Cisco streamlines routing, management



With Cisco's NetFlow Switching, the first packet of traffic goes through several functions task by task ①. The NetFlow cache learns about the flow ② and carries out those tasks at high speed for subsequent packets ③. The information gathered about the flow can then be passed on for network management and planning ④.

SOURCE: CISCO SYSTEMS

Ex. 16 at 19.

The disclosure of NetFlow is consistent with and helps illustrate the disclosure of Kerr. First, "NetFlow collects information about the *first packet* in a stream of data and caches it." *Id.* After receiving that first packet, a "*flow*" of "*several functions*" is then identified, including functions such as "Switching task," "Security task," "Queuing task," and "Accounting task." *Id.* The first packet of traffic "goes through [the] several functions *task by task*." *Id.* The NetFlow system uses "routing and other *information from the first packet*" to handle the remaining packets. *Id.* Thus, once the "cache learns about the flow" of functions, that flow is stored ("*cached*") so that the tasks can be carried out at "high speed" for "*subsequent packets*." *Id.* Information is gathered about the flow for "network management and planning." *State information* is also collected, retrieved, and used for each of the *individual components*

(“functions”), in data structures such as the “Route table,” “Access list,” “Queuing priority,” and “Accounting data.” *Id.*

Thus, to the extent that Kerr is deemed to lack inadequate disclosure of the relevant limitations for claims 1, 15, and 35, the combination of Kerr with NetFlow clearly makes up for any such perceived deficiency.

4. Kerr in View of RFC 1825 and RFC 1829 Renders Obvious Claims 1, 15, and 35 Under § 103

The specification RFC 1825 (“Security Architecture for the Internet Protocol”) (Ex. 26, “RFC 1825”) by R. Atkinson was published in August 1995. The specification RFC 1829 (Ex. 27, “The ESP DES-CBC Transform”) by P. Karn *et al.* was also published in August 1995. Neither was considered during the prosecution of the ‘163 patent.

If certain aspects recited in claims 1, 15, and 35 of the ‘163 patent are not deemed to be disclosed, inherent, or obvious over Kerr alone, then the inclusion of those aspects certainly would be obvious over Kerr in view of RFC 1825 and RFC 1829, under 35 U.S.C. § 103.

It was obvious to supplement the teachings of Kerr with RFC 1825 because Kerr applies “encryption” to “IP” (Internet Protocol) packets, and RFC 1825 (“Security Architecture for the Internet Protocol”) “describes the security mechanisms for IP version 4 (IPv4) and IP version 6 (IPv6) including “encryption.” Ex. 15 (Kerr) at 3:5 (“IP (internet protocol)”), 4:30-31; Ex. 26 (RFC 1825) at 1. It was obvious to supplement the teachings of Kerr and RFC 1825 with RFC 1829, because RFC 1829 teaches an encryption algorithm which “MUST” be supported as part of the RFC 1825 “Security Architecture.” Ex. 26 (RFC 1825) at 10 (the encryption operation “MUST support the use of the Data Encryption Standard (DES) in Cipher-Block Chaining (CBC) Mode”), 21 (citing “RFC 1829”: “The ESP DES-CBC Transform”).

(a) Claim 1

i. “a plurality of components”

Claim 1 recites in pertinent part: “providing a plurality of components, each component being a software routine for converting data with an input format into data with an output format.” Under Implicit’s apparent claim constructions, Kerr in view of RFC 1825 and 1829 renders obvious this element.

As part of its “Security Architecture for the Internet Protocol,” RFC 1825 teaches distinct operations for “Encryption” and “Authentication”—either or both of which may be applied to a packet. Ex. 26 at 1, 3-5, 8-9. The encryption operation is detailed in “RFC 1827” (“IP Encapsulating Security Payload”) and the authentication operation is detailed in “RFC 1826” (“IP Authentication Header”). *See id.* at 3-4, 8-9, 19.

Kerr already teaches the “encryption” of packets in a particular flow. Ex. 15 at 4:30-41. Moreover, it was obvious for Kerr to support encryption and authentication operations as suggested by RFC 1825, both in order to comply with the “Security Architecture for the Internet Protocol,” and also to obtain the security advantages of encryption and authentication detailed by RFC 1825. *E.g.*, Ex. 26 at 1 (“Authentication”: “knowing that the data received is the same as the data that was sent and that the claimed sender is in fact the actual sender.”; “Encryption: “A mechanism that is commonly used to provide confidentiality.”).

Because encryption and authentication are distinct operations which would not necessarily be applied to the same flow, it was obvious to implement them using distinct components, and therefore also obvious to implement them as distinct software routines. *See id.* at 3-5, 8-9.

A component which performed encryption on a packet in the manner described by RFC 1825 would add an “IP Encapsulating Security Payload . . . header” to the packet. Ex. 26 (RFC

1825) at 3. Under Implicit’s apparent claim constructions, this would comprise “converting data with an input format into data with an output format.”

A component which performed authentication on a packet in the manner described by RFC 1825 would add an “IP Authentication Header” to the packet. *Id.* Under Implicit’s apparent claim constructions, this would comprise “converting data with an input format into data with an output format.”

Thus, considering merely these two obvious components alone, they would comprise “a plurality of components, each component being a software routine for converting data with an input format into data with an output format” under Implicit’s apparent claim constructions.

ii. “state information”

Claim 1 finally recites in pertinent part: “for each of a plurality of packets of the message in sequence, for each of a plurality of components in the identified non-predefined sequence, **retrieving** state information relating to performing the processing of the component with the previous packet of the message; **performing** the processing of the identified component with the packet and the retrieved state information; and **storing** state information relating to the processing of the component with the packet for use when processing the next packet of the message.” Under Implicit’s apparent claim constructions, Kerr in view of RFC 1825 and RFC 1829 renders obvious these “state information” elements.

As explained above, it was obvious to provide separate components for encryption and authentication.

RFC 1825 explains that various forms of state information would be maintained by these components, including, *e.g.*, “Key(s) used with the authentication algorithm”; “Key(s) used with the encryption algorithm”; “Authentication algorithm and algorithm mode being used”; “Encryption algorithm, algorithm mode, and transform being used”; “cryptographic

synchronisation or initialisation vector field for the encryption algorithm”; “Lifetime of the key or time when key change should occur”; and “Lifetime of [the] Security Association.” Ex. 9 (RFC 1825) at 5-6. Obvious implementations to maintain this state information would read on this claim element, under Implicit’s apparent claim constructions. For example, both the encryption and authentication component would maintain “Lifetime of the key or time when key change should occur.” *See id.* Maintaining a “Lifetime of the key” (as opposed to maintaining “time when key change should occur”) at least renders obvious a countdown implementation wherein the remaining lifetime is updated with each invocation of the component.

Additionally, regarding the encryption component in particular, an obvious implementation of its encryption technique would read on these claim elements in still another manner, under Implicit’s apparent claim constructions. RFC 1825 explains that the encryption algorithm of RFC 1829 “MUST” be supported for encrypting packets. Ex. 26 (RFC 1825) at 10 (“the IP Encapsulating Security Payload MUST support the use of the Data Encryption Standard (DES) in Cipher-Block Chaining (CBC) Mode”), 21 (citing RFC 1829: “The ESP DES-CBC Transform”). RFC 1829 explains that in order to apply its encryption technique, “an Initialization Vector (IV) that is eight octets in length” must be placed in “[e]ach datagram” to be encrypted (*i.e.*, in each packet). Ex. 27 (RFC 1829) at 1. RFC 1829 further explains that while the “method for selection of IV values is implementation dependent,” a “common acceptable technique is simply a counter, beginning with a random chosen value.” *Id.* It was therefore obvious to apply this counter technique to the encryption component of Kerr. Doing so would clearly entail, for each packet: *e.g.*, retrieving the previous counter value, applying it the packet it, incrementing the counter value, and storing it for use when encrypting the next packet. Under

Implicit's apparent claim constructions, this obvious implementation would read on these claim elements.

(b) Claim 15

i. "dynamically identifying a non-predefined sequence"

Claim 15 recites in pertinent part: "dynamically identifying a non-predefined sequence of components for processing each message based on the first packet of the message so that subsequent packets of the message can be processed without re-identifying the components." Under Implicit's apparent claim constructions, Kerr in view of RFC 1825 and RFC 1829 renders obvious this element.

As explained above, one such obvious sequence would comprise a stateful encryption compression component and a stateful authentication component. *See* Claim 1 above.

ii. "state information"

Claim 15 finally recites in pertinent part: "for each packet of each message, performing the processing of the identified non-predefined sequence of components of the message wherein state information generated by performing the processing of a component for a packet is available to the component when the component processes the next packet of the message." Under Implicit's apparent claim constructions, Kerr in view of RFC 1825 and RFC 1829 renders obvious this element. *See* Claim 1(ii) above (showing similar "state information" element).

(c) Claim 35

i. "dynamically identifying a . . . non-predefined sequence"

Claim 35 recites in pertinent part: "dynamically identifying a message-specific non-predefined sequence of components for processing the packets of each message upon receiving the first packet of the message." Under Implicit's apparent claim constructions, Kerr in view of RFC 1825 and RFC 1829 renders obvious this element.

As explained above, one such obvious sequence would comprise a stateful encryption compression component and a stateful authentication component. *See* Claim 1 above.

ii. “state information”

Claim 35 finally recites in pertinent part: “for each packet of the message, invoking the identified non-predefined sequence of components in sequence to perform the processing of each component for the packet wherein each component saves message-specific state information so that that component can use the save message-specific state information when that component performs its processing on the next packet of the message.” Under Implicit’s apparent claim constructions, Kerr in view of RFC 1825 and RFC 1829 renders obvious this element. *See* Claim 1(ii) above (showing similar “state information” element).

5. Kerr in View of Bellare97 and Bellare95 Renders Obvious Claims 1, 15, and 35 Under § 103

The article “A Concrete Security Treatment of Symmetric Encryption” (Ex. 17, “Bellare97”) by M. Bellare *et al.* was published in 1997. The article “XOR MACs: New Methods for Message Authentication Using Finite Pseudorandom Functions” (Ex. 18, “Bellare95”) by M. Bellare et al. was published in 1995. Neither was considered during the prosecution of the ‘163 patent.

If certain aspects recited in claims 1, 15, and 35 of the ‘163 patent are not deemed to be disclosed, inherent, or obvious over Kerr, then the inclusion of those aspects certainly would be obvious over Kerr in view of Bellare97 and Bellare95, under 35 U.S.C. § 103.

It was obvious to supplement the teachings of Kerr with Bellare97, because Kerr discloses “encryption” of the packets of a flow, and Bellare97 discloses a specific encryption algorithm that could be used. Ex. 15 at 4:30-31. It was obvious to supplement the teachings of

Kerr and Bellare⁹⁷ with Bellare⁹⁵, because Bellare⁹⁵ teaches a similar authentication algorithm which could also be applied to the packets of a flow.

(a) *Claim 1*

i. “a plurality of components”

Claim 1 recites in pertinent part: “providing a plurality of components, each component being a software routine for converting data with an input format into data with an output format.” Under Implicit’s apparent claim constructions, Kerr in view of Bellare⁹⁷ and Bellare⁹⁵ renders obvious this element.

As explained above, it was obvious that each of the distinct operations that Kerr performs on the packets of a flow (including its encryption operation) would be provided by a distinct software routine. *See* Section V.B.2 (Kerr 103) at Claim 1(ii) (showing same element).

Bellare⁹⁵ teaches another operation that would advantageous to apply to the packets of a flow—“Authentication”—and it was obvious that this operation by provided by a distinct software routine as well. Ex. 18 at 1 (“A message authentication scheme enables two parties sharing a key . . . to authenticate their transmissions. This is one of the most widely used cryptographic primitives,” and “as security concerns grow,” “it may become even more so”).

Because Kerr teaches encryption is selectively applied to specific flows, it was obvious to treat authentication in the same manner. *E.g.*, Ex. 15 at 4:30-31.

An encryption component would, of course, convert packet data from an unencrypted format to an encrypted format, and under Implicit’s apparent claim constructions, this would comprise “converting data with an input format into data with an output format.”

An authentication component would insert an extra field in the packet containing a “message authentication code.” Ex. 18 at 1 (“Message authentication is usually accomplished by including with each transmitted message *M* a short string, called its ‘message authentication

code' (MAC) or 'signature'). Under Implicit's apparent claim constructions, this would also comprise "converting data with an input format into data with an output format."

Thus, considering merely these two obvious components alone, they would comprise "a plurality of components, each component being a software routine for converting data with an input format into data with an output format," under Implicit's apparent claim constructions

ii. "state information"

Claim 1 finally recites in pertinent part: "for each of a plurality of packets of the message in sequence, for each of a plurality of components in the identified non-predefined sequence, **retrieving** state information relating to performing the processing of the component with the previous packet of the message; **performing** the processing of the identified component with the packet and the retrieved state information; and **storing** state information relating to the processing of the component with the packet for use when processing the next packet of the message." Under Implicit's apparent claim constructions, Kerr in view of Bellare97 and Bellare95 renders obvious this element.

Bellare97 teaches "*stateful* encryption schemes, in which the ciphertext is a function of some information, such as a counter, maintained by the encrypting party and updated with each encryption." Ex. 17 at 397 (emphasis in original). In its analysis of "some classic symmetric encryption schemes," Bellare97 concludes that a particular stateful scheme ("stateful XOR, based on a finite PRF") "has the best security." *Id.* at 396. "For the stateful XOR scheme we show that . . . this scheme is about as good a scheme as one can possibly hope to get." *Id.* It was therefore obvious to employ such a stateful algorithm in an encryption component of Kerr, particularly since Kerr does not specify a particular encryption algorithm.

Bellare95 teaches “stateful” authentication algorithms in which “the signer maintains information, in our case a counter, which he updates each time a message is signed.” Ex. 18 at

16. In more detail:

In a stateful message authentication scheme, the signer maintains state across consecutive signing requests. (For example, in our counter-based scheme the signer maintains a message counter.) In such a case the signing algorithm can be thought of as taking an additional input—the “current” state *C*, of the signer—and returning an additional output—the signer’s next state.

Id. at 21. Bellare95 analyzes both stateless (“Randomized XOR”) and stateful (“Counter-Based XOR”) authentication algorithms, and observes that “[t]he gain” of the stateful, counter-based algorithm “is greater security.” *Id.* at 22-25 (analysis of stateless), 25-27 (analysis of stateful, counter-based). It was therefore obvious to employ such a stateful algorithm in an authentication component of Kerr.

The counter used for both stateful encryption and stateful authentication would comprise “state information” which is **retrieved** each time another packet is to be encrypted or authenticated, used to **perform** the encryption or authentication, and updated and **stored** so it may be used when encrypting or authenticating the next packet.

(b) Claim 15

i. “dynamically identifying a non-predefined sequence”

Claim 15 recites in pertinent part: “dynamically identifying a non-predefined sequence of components for processing each message based on the first packet of the message so that subsequent packets of the message can be processed without re-identifying the components.”

Under Implicit’s apparent claim constructions, Kerr in view of Bellare97 and Bellare95 renders obvious this element.

As explained above, one such obvious sequence would comprise a stateful encryption component and a stateful authentication component. *See* Claim 1 above.

ii. “state information”

Claim 15 finally recites in pertinent part: “for each packet of each message, performing the processing of the identified non-predefined sequence of components of the message wherein state information generated by performing the processing of a component for a packet is available to the component when the component processes the next packet of the message.” Under Implicit’s apparent claim constructions, Kerr in view of Bellare97 and Bellare95 renders obvious this element.

As explained above, it was obvious for the encryption and authentication components to each employ a stateful algorithm wherein a counter would comprise “state information” which is updated during every encryption or authentication operation. *See* Claim 1(ii) above. This “state information” is thus generated by performing the processing of the component, and is available to the component when processing the next packet of the message. *See id.*

(c) Claim 35

i. “dynamically identifying a . . . non-predefined sequence”

Claim 35 recites in pertinent part: “dynamically identifying a message-specific non-predefined sequence of components for processing the packets of each message upon receiving the first packet of the message.” Under Implicit’s apparent claim constructions, Kerr in view of Bellare97 and Bellare95 renders obvious this element.

As explained above, it was obvious that one such sequence would comprise a stateful encryption component and a stateful authentication component. *See* Claim 1 above.

ii. “state information”

Claim 35 finally recites in pertinent part: “for each packet of the message, invoking the identified non-predefined sequence of components in sequence to perform the processing of each component for the packet wherein each component saves message-specific state information so that that component can use the save message-specific state information when that component performs its processing on the next packet of the message.” Under Implicit’s apparent claim constructions, Kerr in view of Bellare97 and Bellare95 renders obvious this element.

As explained above, it was obvious for the encryption and authentication components to each employ a stateful algorithm wherein a counter would comprise “state information” which is updated during every encryption or authentication operation. *See* Claim 1(ii) above. This “state information” is thus generated by performing the processing of the component, and is available to the component when processing the next packet of the message. *See id.*

6. Kerr in View of IBM96 Renders Obvious Claims 1, 15, and 35 Under § 103

The book “Local Area Network Concepts and Products: Routers and Gateways” (Ex. 19, “IBM96”) was published by IBM in May 1996. IBM96 was not considered during prosecution of the ‘163 patent.

If certain aspects recited in claims 1, 15, and 35 of the ‘163 patent are not deemed to be disclosed, inherent, or obvious over Kerr alone, then the inclusion of those aspects certainly would be obvious over Kerr in view of IBM96, under 35 U.S.C. § 103.

It was obvious to supplement the teachings of Kerr with IBM96 because Kerr teaches a flow-based architecture for routing devices, and IBM96 teaches features which would have been typical of routing devices of the time period.

(a) Claim 1

i. “a plurality of components”

Claim 1 recites in pertinent part: “providing a plurality of components, each component being a software routine for converting data with an input format into data with an output format.” Under Implicit’s apparent claim constructions, Kerr in view of IBM96 renders obvious this element.

During the pertinent time period, it was commonplace for routers to perform compression on certain traffic being routed through them. This is repeatedly confirmed by IBM96. For example, the “IBM 2210 Nways Multiprotocol Router” could perform “Data Compression over Point-to-Point Protocol” using the “LZ77” compression algorithm. Ex. 19 at 84, 95-96. As another example, IBM96 lists “Data compression” as one of the “Advantages” of its “IBM AnyNet Product Family,” explaining that data compression “reduces the amount of data being exchanged between partners, thus improving response time and reducing traffic over the network.” *Id.* at 33. Similarly, IBM96 lists “Data compression” one of the “Benefits” of the “2217 Nways Multiprotocol Concentrator” product, explains data compression “[p]rovides higher data rates and improves response times at a lower cost.” *Id.* at 200-201.

In view of these various benefits of data compression, it was obvious that in addition to supporting operations such as encryption and packet rewrite, Kerr should also support compression. Because Kerr teaches encryption is selectively applied to specific flows, it was obvious to treat compression in the same manner. *E.g.*, Ex. 15 at 4:30-31.

As explained above, it was obvious that each of the distinct operations that Kerr performs on the packets of a flow (including its encryption operation) would be provided by a distinct software routine. *See* Section V.B.2 (Kerr 103) at Claim 1(ii) (showing same element). Likewise, it was obvious compression would be provided by a distinct software routine.

Since compressed data has a different format from uncompressed data, such a compression component would “convert[] data with an input format into data with an output format.” As combined with other component(s) from Kerr (*e.g.*, which perform stateful encryption), such a compression component would comprise “a plurality of components” which read on this claim element.

ii. “state information”

Claim 1 finally recites in pertinent part: “for each of a plurality of packets of the message in sequence, for each of a plurality of components in the identified non-predefined sequence, **retrieving** state information relating to performing the processing of the component with the previous packet of the message; **performing** the processing of the identified component with the packet and the retrieved state information; and **storing** state information relating to the processing of the component with the packet for use when processing the next packet of the message.” Under Implicit’s apparent claim constructions, Kerr in view of IBM96 renders obvious these “state information” elements.

IBM96 discusses and compares the performance of four specific compression algorithms, the top three of which are all “LZ”-based compression algorithms. *See* Ex. 19 at 95-96 (“LZ77” has compression ratio of “2.08:1”; “Stacker-LZS” a ratio of “1.82:1”; “BSD Compress-LZW” a ratio of “2.235:1”; and “Predictor” a ratio of “1.67:1”). Because the top three algorithms discussed by IBM96 are LZ-based and because the “IBM 2210” router specifically uses the “LZ77” algorithm, an LZ-based algorithm such as LZ77 would have been an obvious choice for a compression component to be added to Kerr. *Id.* at 95-96, 84.

LZ compression algorithms are stateful, and an obvious implementation of them would read on these “state information” claim elements.¹⁶ Maintaining such state information would entail, for each packet: *e.g.*, **retrieving** the state information, using it to **perform** the compression processing, updating it to reflect the data in the most recent packet, and **storing** it so it can be applied to the next packet.

More generally (and not confined to LZ-based algorithms), stateful (“adaptive”) compression algorithms were commonplace at the time, and obvious implementations of them would likewise read on this “state information” claim element.¹⁷

(b) *Claim 15*

i. *“dynamically identifying a non-predefined sequence”*

Claim 15 recites in pertinent part: “dynamically identifying a non-predefined sequence of components for processing each message based on the first packet of the message so that subsequent packets of the message can be processed without re-identifying the components.” Under Implicit’s apparent claim constructions, Kerr in view of IBM96 renders obvious this element.

As explained above, one such obvious sequence would comprise a stateful compression component combined with other component(s) (*e.g.*, a stateful encryption component). *See* Claim 1 above.

¹⁶ *See, e.g.*, Ex. G09 (“Nelson”) (“The Data Compression Book” by Mark Nelson *et al.*) (1995) at 21 (LZ employs an “adaptive” algorithm which maintains state information in form of, *e.g.*, a sliding “4K-byte window” of the most recent data seen, or an incrementally built dictionary based on *all* of the previously seen data), 18-19. This reference is cited in this context solely to help explain IBM96. *See* MPEP § 2205.

¹⁷ *See, e.g.*, Ex. G09 (Nelson) at 18 (“compression research in the last 10 years has concentrated on adaptive models”), 18-19 (including Figures 2.2 and 2.3, showing state information in form of a “Model” which is updated on each new piece of data). This reference is cited in this context solely to help explain IBM96. *See* MPEP § 2205.

ii. “state information”

Claim 15 finally recites in pertinent part: “for each packet of each message, performing the processing of the identified non-predefined sequence of components of the message wherein state information generated by performing the processing of a component for a packet is available to the component when the component processes the next packet of the message.”

Under Implicit’s apparent claim constructions, Kerr in view of IBM96 renders obvious this element.

As explained above, it was obvious for the compression component to employ a stateful algorithm (such as “LZ77”) whereby state information would be updated during every compression operation. *See* Claim 1(ii) above. This “state information” is thus generated by performing the processing of the component, and is available to the component when processing the next packet of the message. *See id.*

(c) Claim 35

i. “dynamically identifying a . . . non-predefined sequence”

Claim 35 recites in pertinent part: “dynamically identifying a message-specific non-predefined sequence of components for processing the packets of each message upon receiving the first packet of the message.” Under Implicit’s apparent claim constructions, Kerr in view of IBM96 renders obvious this element.

As explained above, one such obvious sequence would comprise a stateful compression component combined with other component(s) (*e.g.*, a stateful encryption component). *See* Claim 1 above.

ii. “state information”

Claim 35 finally recites in pertinent part: “for each packet of the message, invoking the identified non-predefined sequence of components in sequence to perform the processing of each

component for the packet wherein each component saves message-specific state information so that that component can use the save message-specific state information when that component performs its processing on the next packet of the message.” Under Implicit’s apparent claim constructions, Kerr in view of IBM96 renders obvious this element.

As explained above, it was obvious for the compression component to employ a stateful algorithm (such as “LZ77”) whereby state information would be updated during every compression operation. *See* Claim 1(ii) above. This “state information” is thus generated by performing the processing of the component, and is available to the component when processing the next packet of the message. *See id.*

7. Kerr in View of IBM96 and Nelson Renders Obvious Claims 1, 15, and 35 Under § 103

The treatise “The Data Compression Book” (Ex. 5, “Nelson”) by Mark Nelson et al. was published on November 6, 1995. Nelson was not considered during the prosecution of the ‘163 patent.

If certain aspects recited in claims 1, 15, and 35 of the ‘163 patent are not deemed to be disclosed, inherent, or obvious over Kerr in view of IBM96, then the inclusion of those aspects certainly would be obvious over Kerr in view of IBM96 and Nelson, under 35 U.S.C. § 103.

It was obvious to supplement the teachings of Kerr and IBM96 with Nelson, because IBM96 disclose compression operations performed by routers, and Nelson teaches specific compression algorithms which might be used.

(a) Claim 1

i. “a plurality of components”

Claim 1 recites in pertinent part: “providing a plurality of components, each component being a software routine for converting data with an input format into data with an output

format.” Under Implicit’s apparent claim constructions, Kerr in view of IBM96 and Nelson renders obvious this element.

As explained above, one such obvious sequence would comprise a compression component combined with other component(s) (*e.g.*, a stateful encryption component). *See* Section V.B.5 (Kerr+IBM96) at Claim 1 above.

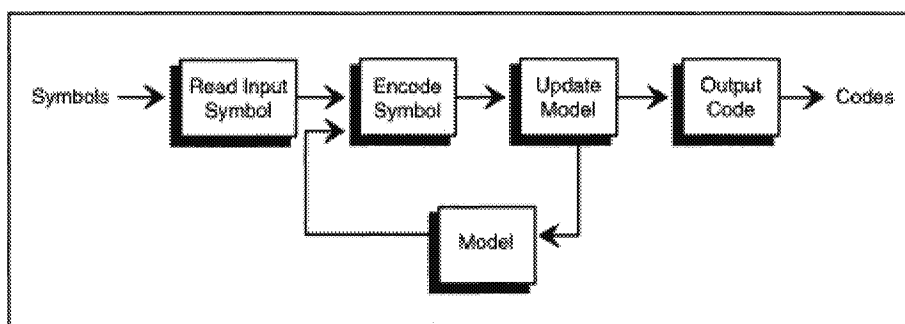
ii. “state information”

Claim 1 finally recites in pertinent part: “for each of a plurality of packets of the message in sequence, for each of a plurality of components in the identified non-predefined sequence, **retrieving** state information relating to performing the processing of the component with the previous packet of the message; **performing** the processing of the identified component with the packet and the retrieved state information; and **storing** state information relating to the processing of the component with the packet for use when processing the next packet of the message.” Under Implicit’s apparent claim constructions, Kerr in view of IBM96 and Nelson renders obvious this “state information” element.

Nelson explains: “Adaptive coding . . . lead[s] to vastly improved compression ratios,” and that “compression research in the last 10 years has concentrated on adaptive models.” Ex. 5 at 8, 18. Adaptive algorithms include such well-known algorithms as “Adaptive Huffman Coding” (chapter 4; *id.* at 75), “Adaptive [Statistical] Modeling” (chapter 6; *id.* at 155), “[Adaptive] Dictionary-Based Compression” (chapter 7; *id.* at 203), and “Sliding Window Compression” (chapter 8; *id.* at 215); and the prominent “LZ” family of compression algorithms (chapter 8 and 9, *id.* at 221, 255). All of these adaptive techniques are lossless, which would be important for accurately transmitting information contained in network packets. *See id.* at 9 (“All of the compression techniques discussed through chapter 9 are ‘lossless’”). In view of the

prominence, lossless nature, and improved compression ratios of adaptive algorithms, use of such an algorithm would have been an obvious choice for a compression component.

Nelson further explains the stateful manner in which adaptive coding operates: “When using an adaptive model, data does not have to be scanned once before coding in order to generate statistics [used to perform compression]. Instead, the statistics are **continually modified as new characters are read in and coded**. The general flow of a program using an adaptive model looks something like that shown in Figure[] 2.2” *Id.* at 18 (emphasis added).



Id. at 19 (Figure 2.2: “General Adaptive Compression,” showing “Update Model” (*i.e.*, update state information) after encoding every piece of data). Nelson explains: “adaptive models start knowing essentially nothing about the data” so “when the program first starts it doesn’t do a very good job of compression.” *Id.* at 19. However, “[m]ost adaptive algorithms tend to adjust quickly to the data stream and will begin turning in respectable compression ratios after only a few thousand bytes.” *Id.*

Thus, an obvious implementation of an adaptive algorithm would entail, for each packet, **retrieving** state information, using it to **perform** the compression processing, updating it to reflect the data in the most recent packet, and **storing** it so it can be applied to the next packet.

More narrowly, IBM96 teaches that its “2210” router employs the “LZ77” compression algorithm, so use of that algorithm in particular would have been an obvious design decision

over IBM96. *See* Ex. 19 (IBM96) at 95-96, 84. Nelson confirms this algorithm was stateful and “adaptive” in the manner described above. *See, e.g.*, Ex. 5 at 21 (“LZ77” maintains a “dictionary” comprised of, *e.g.*, a sliding “4K-byte window” of the most recently seen data).

(b) Claim 15

i. “dynamically identifying a non-predefined sequence”

Claim 15 recites in pertinent part: “dynamically identifying a non-predefined sequence of components for processing each message based on the first packet of the message so that subsequent packets of the message can be processed without re-identifying the components.” Under Implicit’s apparent claim constructions, Kerr in view of IBM96 and Nelson renders obvious this element.

As explained above, one such obvious sequence would comprise a stateful compression component combined with other component(s) (*e.g.*, a stateful encryption component). *See* Claim 1 above.

ii. “state information”

Claim 15 finally recites in pertinent part: “for each packet of each message, performing the processing of the identified non-predefined sequence of components of the message wherein state information generated by performing the processing of a component for a packet is available to the component when the component processes the next packet of the message.” Under Implicit’s apparent claim constructions, Kerr in view of IBM96 and Nelson renders obvious this element.

As explained above, it was obvious for the compression component to employ a stateful, “adaptive” compression algorithm, wherein state information would be updated during every compression operation. *See* Claim 1(ii) above. This “state information” is thus generated by

performing the processing of the component, and is available to the component when processing the next packet of the message. *See id.*

(c) *Claim 35*

i. “*dynamically identifying a . . . non-predefined sequence*”

Claim 35 recites in pertinent part: “dynamically identifying a message-specific non-predefined sequence of components for processing the packets of each message upon receiving the first packet of the message.” Under Implicit’s apparent claim constructions, Kerr in view of IBM96 and Nelson renders obvious this element.

As explained above, one such obvious sequence would comprise a stateful compression component combined with other component(s) (*e.g.*, a stateful encryption and/or authentication component). *See Claim 1 above.*

ii. “*state information*”

Claim 35 finally recites in pertinent part: for each packet of the message, invoking the identified non-predefined sequence of components in sequence to perform the processing of each component for the packet wherein each component saves message-specific state information so that that component can use the save message-specific state information when that component performs its processing on the next packet of the message.” Under Implicit’s apparent claim constructions, Kerr in view of IBM96 and Nelson renders obvious this element.

As explained above, it was obvious for the compression component to employ a stateful, “adaptive” compression algorithm, wherein state information would be updated during every compression operation. *See Claim 1(ii) above.* This “state information” is thus generated by performing the processing of the component, and is available to the component when processing the next packet of the message. *See id.*

8. Kerr in View of RFC 1825, RFC 1829, Bellare97, Bellare95, IBM96, and Nelson Renders Obvious Claims 1, 15, and 35 Under § 103

If certain aspects recited in claims 1, 15, and 35 of the '163 patent are not deemed to be disclosed, inherent, suggested, or obvious over Kerr alone or in combination with the various grounds of rejection presented above, then the inclusion of those aspects certainly would be obvious over Kerr in view of RFC 1825, RFC 1829, Bellare97, Bellare95, IBM96, and Nelson, under 35 U.S.C. § 103, under Implicit's apparent claim constructions.

All of these references have already been combined with Kerr in corresponding sections above, and those sections should be consulted for the detailed manner of applying them to Kerr. This section briefly summarizes that material and shows the collective combination of these references would be obvious as well.

Kerr teaches a general flow-based architecture for router devices which applies, *e.g.*, encryption, packet re-write, and any other "special treatment" to the packets of specific flows. *E.g.*, Ex. 15 at 4:29-60.

RFC 1825 and **Bellare95** confirm the obviousness of employing an additional component for authentication. **IBM96** confirms the obviousness of employing an additional component for compression.

Since Kerr teaches that its various possible operations are applied in a tailored manner to each particular flow (*see id.* at 4:12-20), it was obvious that any two or more of these three types of plugins (encryption, authentication, compression) might be applied to the same flow. This is especially obvious since all three of those operations would be useful for implementing, *e.g.*, a virtual private network across an expensive link, as would be appreciated by one of ordinary skill in the art.

Claims 1, 15, and 35 recite elements regarding "state information."

RFC 1829 and **Bellare97** confirm the obviousness of employing a stateful encryption algorithm which would read on these elements. **Bellare95** confirms the obviousness of employing a stateful authentication algorithm which would read on these elements. **Nelson** confirms the obviousness of employing a stateful compression algorithm which would read on these elements.

Claim 1 recites each component “being a software routine for converting data with an input format into data with an output format.”

Performing encryption on a packet would convert it from an unencrypted to an encrypted format, and likewise performing compression on a packet would convert it from an uncompressed to a compressed format. Both of these operations would read on this “converting data” element, under Implicit’s apparent claim constructions. **Bellare95** confirms that performing authentication on a packet would entail inserting an extra field into the packet, which would also read on this “converting data” element, under Implicit’s apparent claim constructions.

Finally, in addition to the specific plugin components discussed immediately above (encryption, authentication, compression), Kerr discloses a number of other components which would read on the “state information” and/or “format” claim elements of claims 1, 15, and 35, including plugin components for packet rewrite, accounting, and traffic profiling functions. *See* Sections V.B.1 (Kerr 102) and V.B.2 (Kerr 103) above.

Since Kerr teaches that its various possible operations are applied in a tailored manner to each particular flow, it was obvious for any of these various components to be applied to the same flow as well, in addition to (or instead of) any of the encryption, authentication, or compression components discussed immediately above.

In short, there is no aspect of claims 1, 15, and 35 which was not obvious over the prior art and combinations cited herein.

9. Kerr in View of Bellissard Renders Obvious Claims 1, 15, and 35 Under § 103

The article “Dynamic Reconfiguration of Agent-Based Applications” (Ex. 23, “Bellissard”) by Luc Bellissard *et al.* was published by September 10, 1998. Bellissard was not considered during prosecution of the ‘163 patent.

If certain aspects recited in claims 1, 15, and 35 of the ‘163 patent are not deemed to be disclosed, inherent, or obvious over Kerr alone, then the inclusion of those aspects certainly would be obvious over Kerr in view of Bellissard, under 35 U.S.C. § 103.

It was obvious to supplement the teachings of Kerr with Bellissard because Kerr teaches a general flow-based architecture for routers and firewalls (*e.g.*, Ex. 15 at 4:12-48), and Bellissard teaches a technique for enhancing the dynamic extensibility of such an architecture.

(a) Claim 1

i. “dynamically identifying a non-predefined sequence”

Claim 1 recites in pertinent part: “for the first packet of the message, dynamically identifying a non-predefined sequence of components for processing the packets of the message such that the output format of the components of the non-predefined sequence match the input format of the next component in the non-predefined sequence.” Under Implicit’s apparent claim constructions, Kerr in view Bellissard renders obvious this element.

Kerr alone renders obvious this element. *See* Section V.B.2 (Kerr 103) at Claim 1. As applied to Kerr, Bellissard further underscores the “dynamic[.]” nature of the identification, under Implicit’s apparent claim constructions, as explained below.

Bellissard teaches a technique for “dynamically modifying” and “[d]ynamically reconfiguring” an application while the application is *still operating*, without halting the application in order to reconfigure it. Ex. 23 at 1-3. Bellissard explains the motivation for this technique is that “new functionalities” may be “required by the users” at any time:

Reconfiguration is thus an answer to the problems of dynamically modifying the application architecture (both in terms of agent functions and of the sequence of actions to be performed), while the application is operating. This cannot be achieved with current techniques such as configuration of predefined parameters, because it is impossible to predict all the new functionalities that can be required by the users.

Id. at 2.

It was particularly obvious to apply the technique of Bellissard to the router/firewall architecture of Kerr, because a “firewall” is precisely the example chosen by Bellissard of “a typical full-size application” which would “emphasize the benefits of” the Bellissard technique.

Id. at 1; Ex. 15 (Kerr) at 4:45-46 (also “useful for implementing security ‘firewalls’”).

The “dynamic reconfiguration” of technique Bellissard includes performing the following two operations “while the application is operating”: (1) “Modifying the architecture of an application (adding/removing modules, and modifying the interconnection pattern)”; and (2) “Modifying the implementation of a component.” Ex. 23 at 2.

As applied to Kerr, the first operation (“Modifying the architecture of an application” including “adding/removing modules”) would clearly encompass adding or removing certain components of Kerr while the system of Kerr was still operating. *See* Ex. 23 at 2. Bellissard explains “it is impossible to predict all the new functionalities that can be required by users.” Ex. 23 at 2. In the context of the extensible router/firewall architecture of Kerr, providing the required “new functionalities” would typically entail the provision of new Kerr components: *e.g.*, to support a new authentication functionality, a new compression functionality, and so on.

Indeed, Bellissard specifically teaches the insertion of a new “compression” component into a firewall system while it is still operating. Ex. 23 at 2 (“insertion of a compression agent”). Using the Bellissard technique, such new plugins could be “dynamically” added to Kerr while Kerr was still operating—with the advantage that flows could begin to take advantage of the new functionalities immediately, and without disrupting existing flows through the system. *See* Ex. 23 at 1-2.

As applied to Kerr, the second operation (“Modifying the implementation of a component”) would clearly encompass modifying the implementation of a component of Kerr while the system of Kerr was still operating. *See* Ex. 23 at 2. For example, a more efficient, higher-performance implementation might become available for an encryption component, an authentication component, or a compression component, and so on. Using the Bellissard technique, such a component could be “dynamically modified” to employ the new, more efficient implementation while Kerr was still operating—with the advantage that the component could begin to take advantage of the improved implementation immediately, and without disrupting existing flows.

To summarize, the combination of Kerr and Bellissard renders obvious a system in which components of Kerr could be dynamically modified or dynamically added at any moment during runtime—while the system was still operating—and could thereby take advantage of the newly added or modified components. Under Implicit’s apparent claim constructions, such a system would clearly read on “dynamically identifying a non-predefined sequence of components for processing the packets of the message.”

(b) Claim 15

- i. “dynamically identifying a non-predefined sequence”*

Claim 15 recites in pertinent part: “dynamically identifying a non-predefined sequence of components for processing each message based on the first packet of the message so that subsequent packets of the message can be processed without re-identifying the components.” Under Implicit’s apparent claim constructions, Kerr in view of Bellissard renders obvious this element. *See* Claim 1 above.

(c) *Claim 35*

i. “*dynamically identifying a . . . non-predefined sequence*”

Claim 35 recites in pertinent part: “dynamically identifying a message-specific non-predefined sequence of components for processing the packets of each message upon receiving the first packet of the message.” Under Implicit’s apparent claim constructions, Kerr in view of Bellissard renders obvious this element. *See* Claim 1 above.

10. Kerr in View of Fraser Renders Obvious Claims 1, 15, and 35 Under § 103

The publication “DTE Firewalls: Phase Two Measurement and Evaluation Report” (Ex. 24, “Fraser”) by Timothy J. Fraser *et al.* was published by Trusted Information Systems on July 22, 1997. Fraser was not considered during prosecution of the ‘163 patent.

If certain aspects recited in claims 1, 15, and 35 of the ‘163 patent are not deemed to be disclosed, inherent, or obvious over Kerr alone, then the inclusion of those aspects certainly would be obvious over Kerr in view of Fraser, under 35 U.S.C. § 103.

It was obvious to supplement the teachings of Kerr with Fraser because Kerr teaches a general flow-based architecture for routers and firewalls (*e.g.*, Ex. 15 at 4:12-48), and Fraser teaches a technique for enhancing the dynamic configurability of such an architecture.

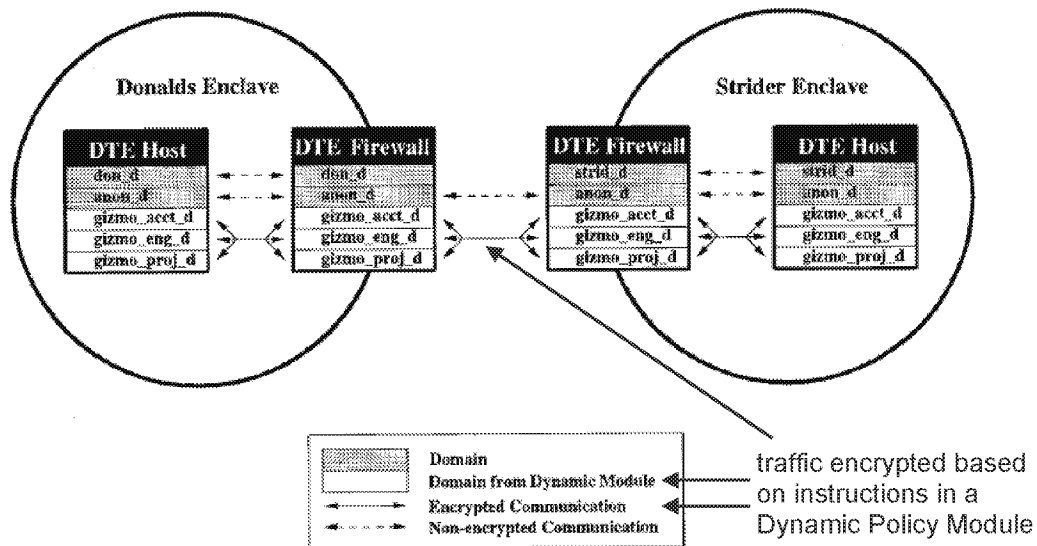
(a) *Claim 1*

i. “*dynamically identifying a non-predefined sequence*”

Claim 1 recites in pertinent part: “for the first packet of the message, dynamically identifying a non-predefined sequence of components for processing the packets of the message such that the output format of the components of the non-predefined sequence match the input format of the next component in the non-predefined sequence.” Under Implicit’s apparent claim constructions, Kerr in view Fraser renders obvious this element.

Kerr alone renders obvious this element. *See* Section V.B.2 (Kerr 103) at Claim 1. As applied to Kerr, Fraser further underscores the “dynamic[]” nature of the identification, under Implicit’s apparent claim constructions, as explained below.

Fraser teaches “Dynamic Policy Modules” which an administrator uses to control the behavior of a firewall: *e.g.*, these modules define which traffic flowing through the firewall should be encrypted, and which network destinations should be accessible to which users. Ex. 24 at 10, 6-7.



Id. at 7 (Figure 3, showing encryption performed according to instructions in “Dynamic Module[s]”; “The transient domains originating in dynamic modules are not shaded.”).

Fraser explains that before Dynamic Policy Modules were introduced, “the primary method” for an administrator to alter a firewall’s “security policy” was “to edit the policy specification and reboot the kernel for the updated policy to take effect.” *Id.* at 8. This approach was “impractical for operational systems,” because “[r]estructuring the policy and rebooting kernels for each change would result in an undesirable and impractical loss of service.” *Id.* at 9.

Dynamic Policy Modules address this “undesirable and impractical” situation by allowing administrators to make minor or major alterations to a firewall’s policies *without* rebooting the device:

The main contribution of dynamic policy module support . . . is increased functionality. As described in section 2.1.2, dynamic policy modules provide administrators with an organized framework for managing policy change. ***Administrators can use dynamic policy modules*** to specify the policy governing new activities and trust relationships. They may add policy support for a new activity or trust relationship to a [firewall] kernel by loading the appropriate module. Similarly, they can remove the support by unloading the module. ***Administrators may load and unload modules as the kernel runs.*** The ability to ***dynamically reconfigure*** a kernel's policy as it runs allows administrators to add and remove policy support for trust relationships without requiring system down-time and the resulting disruption of service availability. This method of policy configuration is ***superior to the [previous] method***, which involved modifying a kernel's base policy description and then ***rebooting*** the kernel.

Id. at 37.

Rather than being narrowly confined to controlling one or two policy options, Dynamic Policy Modules provide a “wide-ranging ability” to change many aspects of a firewall’s policies.

See id. at 19.

Once made available, Dynamic Policy Modules become the primary means for administrators to modify a firewall’s policies: “Dynamic policy modules are the atomic unit of

policy change. Typically, when administrators need to extend a policy to govern a new activity, they will encapsulate the extension in a dynamic policy module.” *Id.* at 12.

It was obvious to apply the Dynamic Policy Modules framework of Fraser to Kerr, in order to provide a more comprehensive framework¹⁸ for avoiding any “undesirable and impractical” need to reboot the Kerr device under any circumstances. *See id.* at 9. Kerr was an especially obvious candidate for this technique, because Fraser uses the technique to control the policies of “firewall[s],” and Kerr teaches an architecture that is “useful for implementing security ‘firewalls’.” *Id.* at 6; Ex. 15 at 4:45-46.

As applied to Kerr, Dynamic Policy Modules would allow an administrator to modify the policies which determine which components are assigned to which flows. *See, e.g.,* Ex. 15 (Kerr) at 4:13-19, 7:47-54. The parallels between the two systems are particularly clear on this point. For example, Fraser’s Dynamic Policy Modules control, *e.g.*, which traffic is encrypted, and Kerr’s policies control, *e.g.*, which flows are encrypted. Ex. 24 at 7, Ex. 15 at 4:12-34.

To summarize, the combination of Kerr and Fraser renders further obvious a system in which the policies determining the identified sequence of plugin components could be dynamically modified or dynamically added at any moment during runtime—while the system was still operating. Under Implicit’s apparent claim constructions, such a system would clearly read on “dynamically identifying a non-predefined sequence of components for processing the packets of the message.”

(b) Claim 15

¹⁸ Kerr already teaches the technique of modifying the system’s configured policies while the system is operating, but Fraser teaches a more comprehensive framework for such a capability, and provides additional detail on how such a framework would be implemented. *See* Ex. B00 (Kerr) at 6:14-16 (“changes in access control lists” cause an existing flow “to be expired”), 8:42-44 (after being “initially configured,” routing device parameters “may be altered by an operator”).

i. “dynamically identifying a non-predefined sequence”

Claim 15 recites in pertinent part: “dynamically identifying a non-predefined sequence of components for processing each message based on the first packet of the message so that subsequent packets of the message can be processed without re-identifying the components.” Under Implicit’s apparent claim constructions, Kerr in view of Fraser renders obvious this element. *See* Claim 1 above.

(c) Claim 35

i. “dynamically identifying a . . . non-predefined sequence”

Claim 35 recites in pertinent part: “dynamically identifying a message-specific non-predefined sequence of components for processing the packets of each message upon receiving the first packet of the message.” Under Implicit’s apparent claim constructions, Kerr in view of Fraser renders obvious this element. *See* Claim 1 above.

11. Kerr in View of RFC 1825, RFC 1829, Bellare97, Bellare95, IBM96, Nelson, Bellissard, and Fraser Renders Obvious Claims 1, 15, and 35 Under § 103

If certain aspects recited in claims 1, 15, and 35 of the ‘163 patent are not deemed to be disclosed, inherent, suggested, or obvious over Kerr alone or in combination with the various grounds of rejection presented above, then the inclusion of those aspects certainly would be obvious over Kerr in view of RFC 1825, RFC 1829, Bellare97, Bellare95, IBM96, Nelson, Bellissard, and Fraser under 35 U.S.C. § 103, under Implicit’s apparent claim constructions.

All of these references have already been combined with Kerr in corresponding sections above, and those sections should be consulted for the detailed manner of applying them to Kerr. This section briefly summarizes that material and shows the collective combination of these references would be obvious as well.

Kerr teaches a general flow-based architecture for router devices which applies, *e.g.*, encryption, packet re-write, and any other “special treatment” to the packets of specific flows. *E.g.*, Ex. 15 at 4:29-60.

RFC 1825 and **Bellare95** confirm the obviousness of employing an additional component for authentication. **IBM96** confirms the obviousness of employing an additional component for compression.

Since Kerr teaches that its various possible operations are applied in a tailored manner to each particular flow (*see id.* at 4:12-20), it was obvious that any two or more of these three types of plugins (encryption, authentication, compression) might be applied to the same flow. This is especially obvious since all three of those operations would be useful for implementing, *e.g.*, a virtual private network across an expensive link, as would be appreciated by one of ordinary skill in the art.

Claims 1, 15, and 35 recite elements regarding “state information.”

RFC 1829 and **Bellare97** confirm the obviousness of employing a stateful encryption algorithm which would read on these elements. **Bellare95** confirms the obviousness of employing a stateful authentication algorithm which would read on these elements. **Nelson** confirms the obviousness of employing a stateful compression algorithm which would read on these elements.

Claim 1 recites each component “being a software routine for converting data with an input format into data with an output format.”

Performing encryption on a packet would convert it from an unencrypted to an encrypted format, and likewise performing compression on a packet would convert it from an uncompressed to a compressed format. Both of these operations would read on this “converting

data” element, under Implicit’s apparent claim constructions. Bellare95 confirms that performing authentication on a packet would entail inserting an extra field into the packet, which would also read on this “converting data” element, under Implicit’s apparent claim constructions.

Finally, in addition to the specific plugin components discussed immediately above (encryption, authentication, compression), Kerr discloses a number of other components which would read on the “state information” and/or “format” claim elements of claims 1, 15, and 35, including plugin components for packet rewrite, accounting, and traffic profiling functions. See Sections V.B.1 (Kerr 102) and V.B.2 (Kerr 103) above.

Since Kerr teaches that its various possible operations are applied in a tailored manner to each particular flow, it was obvious for any of these various components to be applied to the same flow as well, in addition to (or instead of) any of the encryption, authentication, or compression components discussed immediately above.

Claims 1, 15, and 35 recite “dynamically identifying a . . . non-predefined sequence of components.”

Kerr alone makes clear that administrators can make rule-based or policy changes during *runtime*, which falls within the scope of “dynamically identifying a non-predefined sequence of components” under Implicit’s apparent claim construction. *E.g.*, Ex. 15 at 6:14-16 (“changes in access control lists” can occur during an existing “flow,” causing it to “expire”), 8:42-44 (after being “initially configured,” routing device parameters “may be altered by an operator”).

Bellissard teaches dynamically adding new components and modifying existing components while the system is operating. Under Implicit’s apparent claim constructions, both of these techniques would read on these “dynamic[.]” claim elements.

Like Kerr, **Fraser** teaches dynamically configuring firewall policies while the system is operating. It teaches a more comprehensive framework for this capability, and details another manner in which it could be implemented. Under Implicit's apparent claim constructions, such dynamic configuration of policies would read on these "dynamic[]" claim elements.

In short, there is no aspect of claims 1, 15, and 35 which was not obvious over the prior art and combinations cited herein.

12. Kerr in View of Checkpoint and Shwed Renders Obvious Claims 1, 15, and 35 Under § 103

The paper "Checkpoint Firewall-1 White Paper, Version 2.0" (Ex. 20, "Checkpoint") was published by Checkpoint (the maker of the "Firewall-1" product) in September 1995. U.S. Pat. No. 5,835,726 entitled "System for securing the flow of and selectively modifying packets in a computer network," by Shwed et al. (Ex. 21, "Shwed") issued on November 19, 1998 to the assignee Checkpoint Software Technologies Ltd. Neither Checkpoint nor Shwed was considered during the prosecution of the '163 patent.

If certain aspects recited in claims 1, 15, and 35 of the '163 patent are not deemed to be disclosed, inherent, or obvious over Kerr, then the inclusion of those aspects certainly would be obvious over Kerr in view of Checkpoint, and further in view of Shwed, under 35 U.S.C. § 103.

Under Implicit's apparent claim construction, the "dynamically" limitation requires some degree of system configurability, and Kerr duly discloses a fully configurable network security product. However, if Kerr is deemed to lack sufficient disclosure regarding system configurability, combination with Checkpoint and Shwed cures any such deficiency. Checkpoint and Shwed illustrate the fact that network security products such as firewalls have had the ability to arbitrarily add and change rules and policies for years prior to the filing date of the '163 patent. And it would have been obvious to apply the teachings of Checkpoint and Shwed to the

networking technologies in Kerr, to provide yet additional configurability options to address changing security demands in a network environment.

For example, Checkpoint describes the “Inspection Module” of the Firewall-1 product as “generic and *flexible*,” one that is “capable of learning and *understanding any protocol*, as well as *adapting to newly defined protocols and applications*.” Ex. 20 at 20. It goes on to observe that “[t]his capability is achieved by using high-level definitions, *and requires no code changes*.” *Id.* Checkpoint explains that Firewall-1 is not limited to examining header data; it can “extract data from the packet’s *application content* and store it to provide context.” *Id.* at 14. Based on this information, the Firewall-1 “is able to *dynamically* allow and disallow connections as necessary.” *Id.* The Firewall-1 is also able to store and use “*state information* for each session through the gateway,” using a technology known as “Stateful Multi-Layer Inspection.”

Shwed similarly shows the highly configurable nature of the claimed firewall, and even includes a depiction of the “rule base editor”:

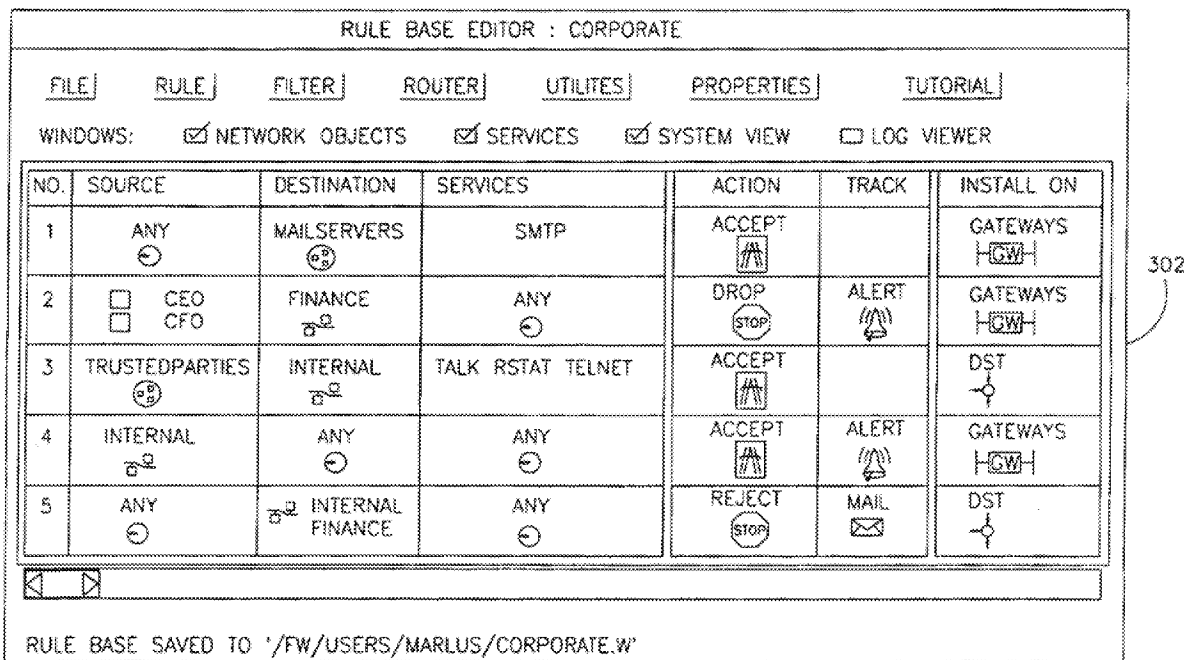


FIG.3/4

Ex. 21 at Fig. 3. Shwed discloses a plurality of “packet filters,” each of which “can *handle changes* in security rules with *great flexibility* as well as handle multiple security rules *without changing the structure of the packet filter itself*.” *Id.* at 6:35-39; *see generally id.* at 5:39 – 8:9. “[E]ach packet filter Like Checkpoint, Shwed also discloses “stateful multi-layer inspection.” *Id.* at 14:55-62.

Thus, to the extent that Kerr is deemed to lack inadequate disclosure of the relevant limitations for claims 1, 15, and 35, the combination of Kerr with Checkpoint and Shwed clearly makes up for any such perceived deficiency.

13. Kerr in View of Dietz Renders Obvious Claims 1, 15, and 35 Under § 103

U.S. Pat. No. 6,651,099 entitled “Method and Apparatus for Monitoring Traffic in a Network” by Russell S. Dietz et al. (Ex. 22, “Dietz”) resulted from a patent application filed on June 30, 1999. Dietz was not considered during the prosecution of the ‘163 patent.

If certain aspects recited in claims 1, 15, and 35 of the '163 patent are not deemed to be disclosed, inherent, or obvious over Kerr, then the inclusion of those aspects certainly would be obvious over Kerr in view of Dietz, under 35 U.S.C. § 103.

For example, Dietz, like Kerr, is expressly described as a “flow”-based system, as illustrated in Figure 3, and thus it would have been obvious to jointly consider their combined teachings:

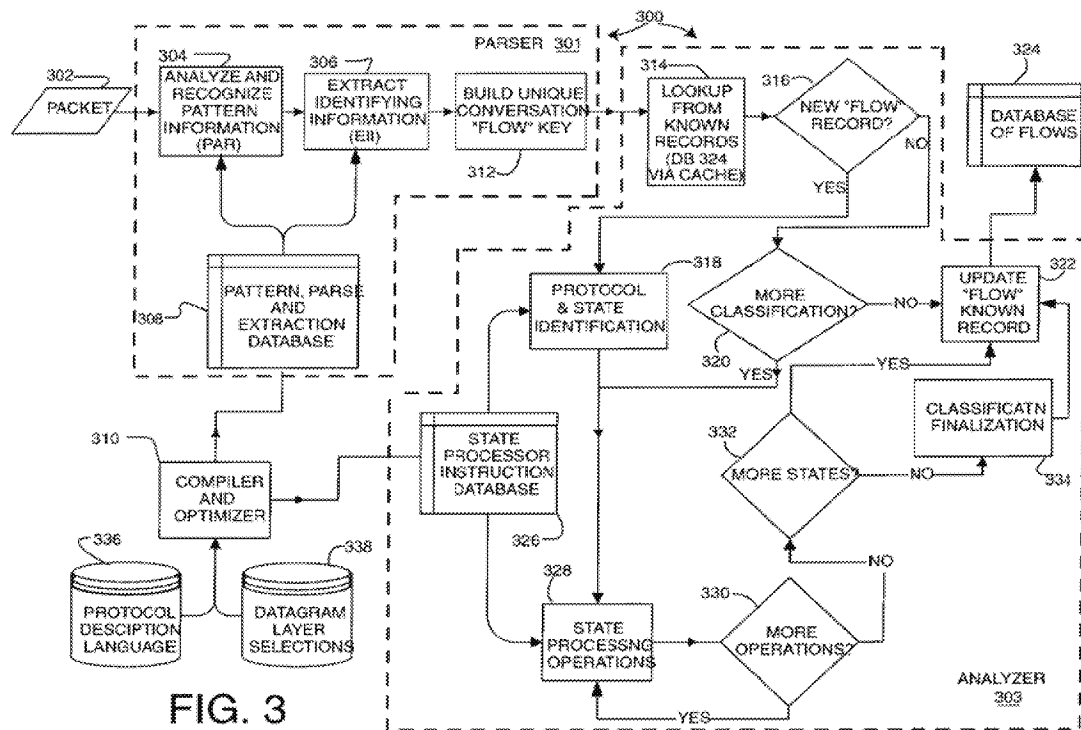


FIG. 3

Ex. 22 at Fig. 3. Dietz also discloses analysis of packets passing through the system “in real time” in order to determine “the application program associated with the conversational flow.” *Id.* at Abstract. In so doing, Dietz looks not only to the “protocol (e.g., http, ftp, H.323, VPN, etc.),” but also “the application/use within the protocol.” *Id.* at 3:30-34. Thus, Dietz is able to provide “a flexible processing system that can be tailored or adapted as new applications enter

the client/server market” so it can classify and respond to flows based on application. *Id.* at 4:45 – 5:9.

Thus, to the extent that Kerr is deemed to lack inadequate disclosure of the relevant limitations for claims 1, 15, and 35, the combination of Kerr with Dietz clearly makes up for any such perceived deficiency.

14. Kerr in view of Pfeifer96 Renders Obvious Claims 1, 15, and 35 Under § 103

If certain aspects recited in claims 1, 15, and 35 of the ‘163 patent are not deemed to be disclosed, inherent, or obvious over Kerr alone, then the inclusion of those aspects certainly would be obvious over Kerr in view of Pfeifer96, under 35 U.S.C. § 103.

(a) Claim 1

i. “processing a message having a series of packets”

Claim 1 recites: “A method in a computer system.” Under Implicit’s apparent claim constructions, Kerr discloses this element.

Kerr discloses a “routing device” which, *e.g.*, maintains data structures called “flow entr[ies]” accessed via a “hash table.” Ex. B10 at Figure 3, 4:9. Under Implicit’s apparent claim constructions, a routing device capable of supporting such data structures would comprise “a computer system.”

ii. “processing a message having a series of packets”

Claim 1 further recites the method is “for processing a message having a sequence of packets.” Under Implicit’s apparent claim constructions, Kerr discloses this element.

Kerr summarizes its invention in part as follows:

The invention provides a method and system for switching in networks responsive to message flow patterns. A message "flow"

is defined to comprise a set of packets to be transmitted between a particular source and a particular destination. When routers in a network identify a new message flow, they determine the proper processing for packets in that message flow and cache that information for that message flow. Thereafter, when routers in a network identify a packet which is part of that message flow, they process that packet according to the proper processing for packets in that message flow. The proper processing may include a determination of a destination port for routing those packets and a determination of whether access control permits routing those packets to their indicated destination.

Id. at 1:48-61.

iii. “a plurality of components”

Claim 1 further recites: “providing a plurality of components, each component being a software routine for converting data with an input format into data with an output format.”

Under Implicit’s apparent claim constructions, Kerr in view of Pfeifer96 renders obvious this element.

The essential idea of the Pfeifer96 system is that users are “mobile” and “nomadic,” but they still wish to receive their electronic communications even when they are away from their “well equipped” offices. Ex. A02 at 19, 1. For example, a user may be away from his office in a location where there is only a telephone—so if a fax is sent to his office, he will surely miss it. *See id.* at 15, 17, 8. Pfeifer96 proposes a general solution to this problem, and this solution relies on three related technologies. First, the system *keeps track* of the user’s current location: users may “register” when they arrive at a location, or they may wear an “Active Badge” which tracks their location automatically. *Id.* at 16, 24. Second, the system keeps track of the specific communications “access devices” in the user’s location which would be available to receive incoming communications. *Id.* at 17. Third and most importantly, the system can dynamically convert incoming communications into an *entirely different medium*, if there is not an access

device in the vicinity capable of receiving the incoming communication in its original form. *Id.* at 17, 21.

For example, if there is telephone but not a fax machine in the user's current location, the system can automatically phone the user and *read him the fax*, by dynamically converting the data from its original source medium (fax) to a medium supported by the access device in the user's vicinity (audio). *Id.* at 8, 20-21.

More specifically, the iPCSS performs this function by transparently observing incoming communications as they enter its "Service Gateways." *Id.* at 17-21. By observing these incoming communications, the system can ascertain: (1) the intended destination of an incoming communication (*i.e.*, the called party); and (2) the communication's source medium (*e.g.*, voice call, videoconference, fax, etc.). *Id.* Knowing these two facts, the system can then determine if there is a device in the called party's "current vicinity" which could accept the source medium. *Id.* at 17. If not, the system can dynamically generate a converter chain for converting the communication to a medium which can be accepted by one of the nearby devices. *Id.* at 17-21.

Pfeifer96 styles this system the "Intelligent Personal Communications Support System (iPCSS)," and its slogan is "*information any time, any place, in any form.*" *Id.* at 2, 14 (emphasis in original).

It was obvious to apply this system of Pfeifer96 to Kerr, so Kerr could assure delivery of incoming communications to users in their *actual* locations.

As an internet router, Kerr is positioned at a natural chokepoint in the network where communication flows to many remote users may be observed transparently as they travel through the router device. *E.g.*, Ex. B10 at 2:56-61; Ex. A02 at 23. And of course as a router, Kerr is ideally suited to re-routing those flows for delivery to the users' *actual* locations, as necessary.

In fact, under Implicit's apparent claim construction, routers can be said to "convert" communications in the course of routing them; *e.g.*, Kerr teaches "encryption" as one of the operations that may be applied to all of the packets of a particular flow. Ex. B10 at 4:4:31.

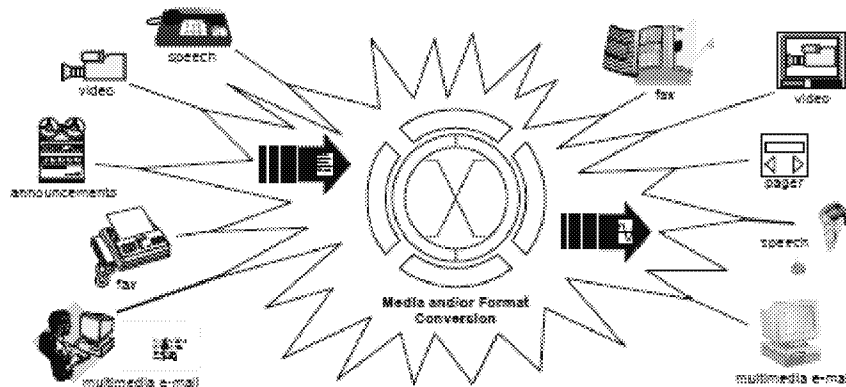
Moreover, Kerr already tracks all incoming and outgoing communications ("flows"), maintaining a separate data structure for each called a "flow entry." *Id.* at 3:4-4-17, 4:28-29, Figure 3.

A message flow **160** consists of a unidirectional stream of packets **150** to be transmitted between particular pairs of transport service access points (thus, network-layer addresses and port numbers). In a broad sense, a message flow **160** thus refers to a communication "circuit" between communication endpoints. In a preferred embodiment, a message flow **160** is defined by a network-layer address for a particular source device **120**, a particular port number at the source device **120**, a network-layer address for a particular destination device **130**, a particular port number at the destination device **130**, and a particular transmission protocol type.

Id. at 2:56-68. Thus, a "flow" is precisely the set of data which would need to be re-routed and converted if the called party was, *e.g.*, not in his office.

Again like the system of Pfeifer96, Kerr is aware of, and tracks, the intended destination of the incoming communication: "the particular destination device **130** [of a flow] is identified by its IP (internet protocol) address." *Id.* at 3:15-17. Thus, Kerr would know precisely where a flow was heading: *e.g.*, to a fax device in the user's office.

Again like the system of Pfeifer96, Kerr tracks the specific *medium* of communication which would be flowing toward the destination: *e.g.*, whether it is "an internet **telephone protocol**, or an internet **video protocol** such as the "CUSeeMe" protocol; these protocols are known in the art of networking." *Id.* at 3:13-15 (emphasis added). And in fact, telephone and video are two of the very mediums which iPCSS automatically detects and converts:



Ex. A02 at 15 (Figure 9: “Priorized media conversion in the iPCSS”).

Thus, considering Kerr in view of Pfeifer96 essentially poses this question to Kerr: *knowing and tracking* all this information about each flow (including its intended destination device, its source medium), and being responsible for routing the flow onward to its intended destination—what should be done if the user is *not in the vicinity of the destination device*? Clearly, an obvious answer is to apply the system of Pfeifer96, whereby a flow can be re-routed and converted for connection to a device at the user’s current location, rather than terminating uselessly at a device in a vacant office.

This obviousness is further heightened by the straightforward compatibility of the two architectures: the one would fit into the other seamlessly.

Kerr teaches a “flow” architecture for processing packets with high performance:

It would therefore be advantageous to provide techniques in which the amount of processing required for any individual packet could be reduced. With inventive techniques described herein, information about message flow patterns is used to identify packets for which processing has already been determined, and therefore to process those packets without having to re-determine the same processing. The amount of processing required for any individual packet is therefore reduced.

Information about message flow patterns would also be valuable for providing information about use of the network, and could be

used for a variety of purposes by network administrators, routing devices, service providers, and users.

Ex. B10 at 1:29-43. When the first packet of a flow arrives, Kerr goes through the somewhat expensive and elaborate process of determining how *all* the packets of that flow should be treated: *e.g.*, whether they should be encrypted, whether their packet headers should re-written, and where they should be routed next. *Id.* at 1:33-35, 4:13-60. It then records all this information about the proper processing for a flow by “build[ing] a new entry in the flow cache” for the flow, so the proper processing does not have to be wastefully and redundantly determined again for subsequent packets of the flow. *Id.* at 4:12-13.

Thus, in a preferred embodiment, the routing device **140** does not separately determine, for each packet **150** in the message flow **160**, the information stored in the entry in the flow cache. Rather, when routing a packet **150** in the message flow **160**, the routing device **140** reads the information from the entry in the flow cache and treats the packet **150** according to the information in the entry in the flow cache.

Ex. B01 at 4:64-5:3.

It is clearly at this moment, when determining the proper processing for the flow, that the system of Kerr would simply query the system of Pfeifer96 to determine if some additional amount of processing would be appropriate for this flow. *E.g.*, if iPCSS determines the user is not at the destination device, it could then (in its usual manner) dynamically generate a suitable converter chain for connecting the flow to a device in the user’s current location. Ex. A02 at 17-21. A reference to the converter chain would then simply be recorded in the flow entry, as part of the processing to be applied to all subsequent the packets of the flow. As explained by Kerr: “It will be clear to those skilled in the art, after perusing this application, that the concept of a message flow is quite broad, and encompasses a wide variety of possible alternatives within the scope and spirit of the invention.” Ex. B10 at 3:21-24.

Returning now more specifically this claim element, this portion of Claim 1 recites: “providing a plurality of components, each component being a software routine for converting data with an input format into data with an output format.”

Though Kerr teaches components of its own for converting data in a particular flow (*e.g.*, to perform encryption or to re-write packet headers), it is not necessary to analyze these because Kerr would incorporate the dynamically generated converter chains of Pfeifer96, which have already been applied to this claim element above. As explained above, such a converter chain would comprise “a plurality of components, each component being a software routine for converting data with an input format into data with an output format.” *See* Section V.A.1 (Pfeifer96 102) at Claim 1(iii).

iv. “dynamically identifying a non-predefined sequence”

Claim 1 further recites: “for the first packet of the message, dynamically identifying a non-predefined sequence of components for processing the packets of the message such that the output format of the components of the non-predefined sequence match the input format of the next component in the non-predefined sequence.” Under Implicit’s apparent claim constructions, Kerr in view of Pfeifer96 renders obvious this element.

When the first packet of a new flow is received, Kerr “builds a new entry in the flow cache” for it, “determines proper treatment of packets . . . in the message flow,” and records that information in the flow entry. *See* Ex. B10 at 4:12-52; Section iii above. As explained above, part of determining the “proper treatment” for packets in the flow would include performing the Pfeifer96 procedure of determining if the flow should be re-routed through a converter chain for delivery to a user at his actual location—and dynamically generating a suitable chain if so. *See* Section iii above. As explained elsewhere above, this Pfeifer96 procedure would read on these claim elements. *See* Section V.A.1 (Pfeifer96 102) at Claim 1(iv).

v. *“selecting individual components”*

Claim 1 further recites that “dynamically identifying includes selecting individual components to create the non-predefined sequence of components after the first packet is received.” Under Implicit’s apparent claim constructions, Pfeifer96 discloses this element.

As explained above, part of determining the “proper treatment” for packets in the flow would include performing the Pfeifer96 procedure of determining if the flow should be re-routed through a converter chain for delivery to a user at his actual location. *See* Section iii above. As explained above, this Pfeifer96 procedure would read on these claim elements. *See* Section V.A.1 (Pfeifer96 102) at Claim 1(iv).

vi. *“storing an indication of . . . the identified components”*

Claim 1 further recites “storing an indication of each of the identified components so that the non-predefined sequence does not need to be re-identified for subsequent packets of the message.” Under Implicit’s apparent claim constructions, Pfeifer96 discloses this element.

As explained above, it was obvious that a reference to the dynamically generated converter chain for a flow would be stored in the flow’s entry. *See* Section iii above. Kerr explains the purpose of its flow entries and their implementation as follows:

It would therefore be advantageous to provide techniques in which the amount of processing required for any individual packet could be reduced. With inventive techniques described herein, information about message flow patterns is used to identify packets for which processing has already been determined, and therefore to process those packets **without having to re-determine** the same processing. The amount of processing required for any individual packet is therefore reduced

Thus, in a preferred embodiment, the routing device 140 **does not separately determine, for each packet** 150 in the message flow 160, the information stored in the entry in the flow cache. Rather, when routing a packet 150 in the message flow 160, the routing device 140 **reads the information from the entry in the flow**

cache and treats the packet 150 according to the information in the entry in the flow cache.

Ex. B10 at 1:29-37, 4:64-5-4 (emphasis added).

vii. “state information”

Claim 1 further recites “for each of a plurality of packets of the message in sequence, for each of a plurality of components in the identified non-predefined sequence, **retrieving** state information relating to performing the processing of the component with the previous packet of the message; **performing** the processing of the identified component with the packet and the retrieved state information; and **storing** state information relating to the processing of the component with the packet for use when processing the next packet of the message.” Under Implicit’s apparent claim constructions, Pfeifer96 discloses this “state information” element.

As explained above, the system of Kerr in view of Pfeifer96 would incorporate the dynamically generated converter chains of Pfeifer96, and these would read on these claim elements. *See* Section iii; Section V.A.1 (Pfeifer96 103) at Claim 1(vii).

(b) Claims 15 and 35

The combination of Kerr and Pfeifer would also render obvious claims 15 and 35, for the reasons set forth immediately above as to claim 1, and in light of the fact that both Kerr and Pfeifer separately disclose every limitation of claims 15 and 35 for the reasons set forth in Section V.A.1 and V.B.1.

C. Decasper98 (Exhibit 25)

The article “Router Plugins: A Software Architecture for Next Generation Routers” by Dan Decasper *et al.* (“Decasper98”) was published in October 1998. Decasper98 was not considered during prosecution of the ‘163 patent.

1. Decasper98 Anticipates Claims 1, 15, and 35 Under § 102(a), (b)

(a) *Claim 1*

i. “A method . . . for processing a message”

Claim 1 recites: “A method in a computer system for processing a message having a sequence of packets, the method comprising” Under Implicit’s apparent claim constructions, Decasper98 discloses this element.

Decasper98 teaches “an extensible and modular software architecture for high-performance . . . routers” which “allows code modules called plugins to be dynamically loaded into the kernel and configured at run time.” Ex. 25 at 11. Under Implicit’s apparent claim constructions, a router capable of implementing such a software architecture would comprise “a computer system.”

Claim 1 further recites the method is “for processing a message having a sequence of packets.” Decasper98 explains: “it is very important to be able to quickly and efficiently classify packets into flows, and to apply different policies to different flows; these are both things that our architecture excels at doing.” Ex. 25 at 2. Flows may represent “longer lived packet streams”:

Because the deployment of multimedia data sources and applications (e.g. real-time audio/video) will produce longer lived **packet streams** with **more packets per session** than is common in today’s environment, an integrated services router architecture should support the notion of flows and build upon it.

Id. at 3. A flow is defined as a group of packets which satisfy a specific filter. *See id.* at 3 (“Sets of flows are specified using *filters* Filters can also match individual end-to-end application flows”). *Id.* at 3. A flow would comprise a “message” under Implicit’s apparent claim constructions. *See* Section IV.C.

ii. “a plurality of components”

Claim 1 further recites: “providing a plurality of components, each component being a software routine for converting data with an input format into data with an output format.”

Under Implicit’s apparent claim constructions, Decasper98 discloses this element.

Decasper98 teaches that “[o]ne of the novel features of our design is the ability to bind different plugins to individual flows.” *Id.* at 1.

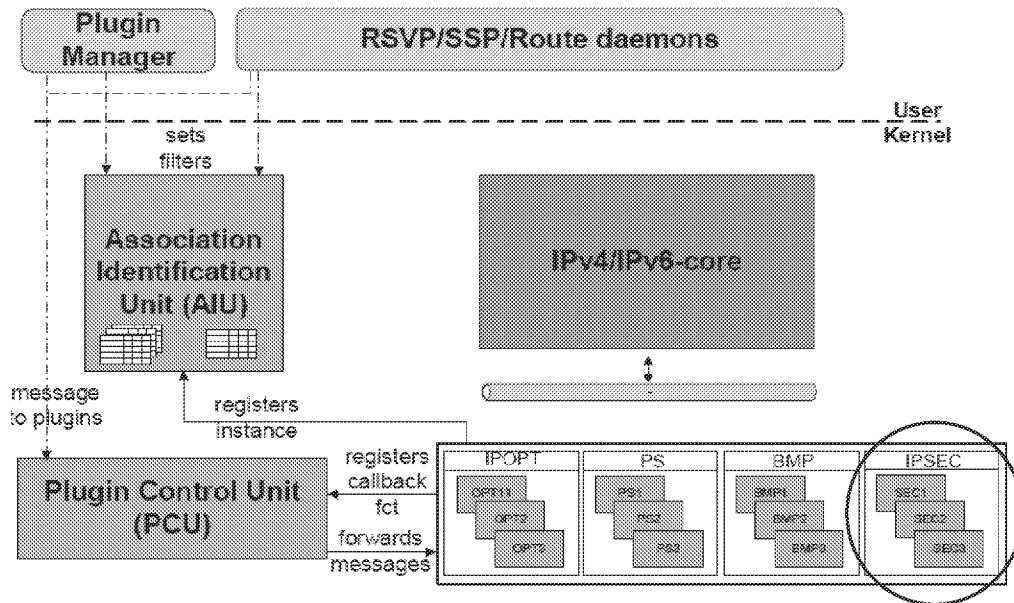
These “plugins” are software routines under Implicit’s apparent claim constructions. *See id.* at 2 (“plugins are kernel software modules that are . . . responsible for performing certain functions on specified network flows.”).

These “plugins” also convert data with an input format into data with an output format. Under Implicit’s apparent claim construction, all of the plugins taught by Decasper98 would read on these claim elements, including “plugins implementing IPv6 options, plugins for packet scheduling . . . plugins for IP security . . . a routing plugin, a statistics gathering plugin . . . a firewall plugin,” and so on. Ex. 25 at 4. Additional examples are set forth below.

(a) IP security components

As a first group of examples, Decasper98 teaches multiple plugin components for implementing “IP security,” which refers to IP security standards including RFC 1825 (“Security Architecture for the Internet Protocol”), RFC 1826 (“IP Authentication Header”), RFC 1827 (“IP Encapsulating Security Payload”), and RFC 1829 (“The ESP DES-CBC Transform”). *See* Ex. 25 at 2 (“plugins for IP Security” citing footnote “[2]”), 12 (footnote “[2]” citing “RFC 1825”); Ex. 26 (RFC 1825) at 3 (citing “two specific headers . . . used to provide security in IPv4 and

IPv6” which may be added to a packet: an authentication header (detailed in RFC 1826) and an encryption header (detailed in RFC 1827)).¹⁹



Ex. 25 at 4 (Figure 2, showing multiple “IPSEC” plugins: “SEC1 SEC2 SEC3”).

As taught by both Decasper98 and RFC 1825, these IP security operations are used to form virtual private networks. Compare Ex. 25 at 5 (“IP security processing has to be done if the system is configured as entry point into a virtual private network”); Ex. 26 (RFC 1825) at 4 (“building private virtual networks across an untrusted backbone”).

A plugin component which performed encryption on a packet in the manner described by RFC 1825 would add an “IP Encapsulating Security Payload . . . header” to the packet. See Ex. 26 (RFC 1825) at 3. Under Implicit’s apparent claim constructions, this would comprise “converting data with an input format into data with an output format.”

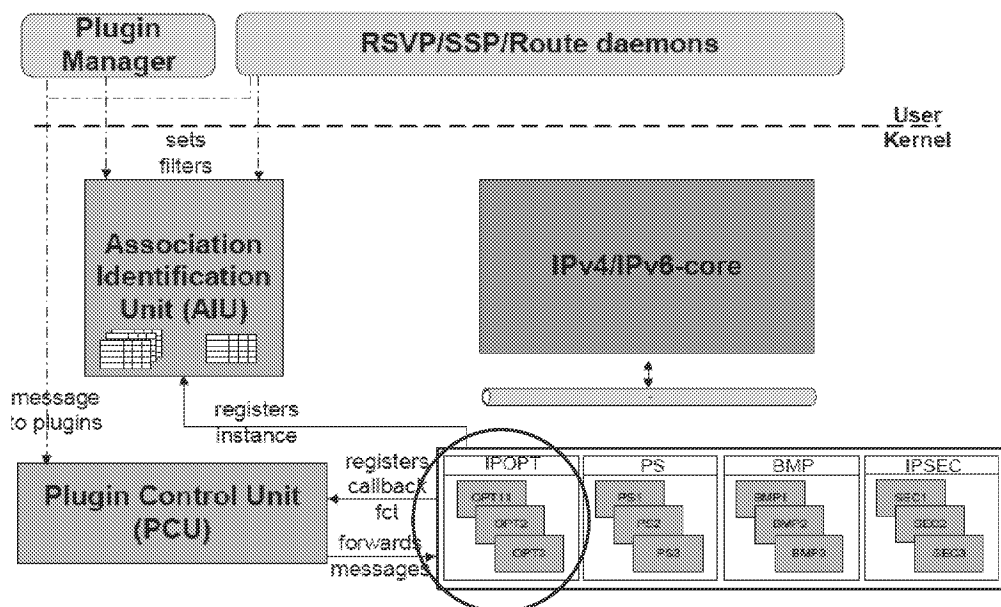
A plugin component which performed authentication on a packet in the manner described by RFC 1825 would add an “IP Authentication Header” to the packet. *Id.* Under Implicit’s

¹⁹ RFC 1825 is relied on by Decasper98, and is cited throughout this section solely to help explain Decasper98. See MPEP § 2205; Ex. 25 at 2, 7, 12 (Decasper98 citations to RFC 1825).

apparent claim constructions, this would comprise “converting data with an input format into data with an output format.”

(b) IPv6 options components

As a second group of examples, Decasper98 teaches multiple “plugins implementing IPv6 options.” Ex. 25 at 4. *See also id.* at 6 (some plugins “can be very simple (e.g., a dozen lines of code for an IP option plugin)”) (further clarifying there would be multiple such option plugins).



Id. at 4 (Figure 2, showing multiple “IPOPT” plugins: “OPT1 OPT2 OPT3”).

Components which perform such IPv6 option processing would add or remove IPv6 option headers in the course of processing the packet.²⁰ Under Implicit’s apparent claim

²⁰ *See, e.g.*, Ex. 29 (“IPv6: The New Internet Protocol” by Christian Huitema) (1997) at 15 (figure showing “Daisy Chain” of IPv6 “extension headers”), 25 (“recommended order” of IPv6 extension headers includes “2. Hop-by-Hop options header 3. Destination options header”; “‘onion-peeling’ procedure” for extension headers wherein “[e]ach successive layer would be processed in turn, just like removing each layer of an onion in turn”). This reference is cited in this context solely to help explain Decasper98, which refers to IPv6 options. *See* MPEP § 2205.

constructions, this would comprise “converting data with an input format into data with an output format.”

iii. “dynamically identifying a non-predefined sequence”

Claim 1 further recites: “for the first packet of the message, dynamically identifying a non-predefined sequence of components for processing the packets of the message such that the output format of the components of the non-predefined sequence match the input format of the next component in the non-predefined sequence, wherein dynamically identifying includes selecting individual components to create the non-predefined sequence of components after the first packet is received.” Under Implicit’s apparent claim constructions, Decasper98 discloses this element—and it does so in at least two distinct manners, as will be explained below.

When the first packet of a new flow arrives, Decasper98 performs an expensive series of filter operations to determine the correct sequence of plugin components to be applied to the flow. *See* Ex. 25 at 5-6 (“The processing of the first packet of a new flow . . . involves n filter table lookups to create a single entry in the flow table for the new flow.”). This expensive series of filter operations does not need to be repeated for subsequent packets of the flow, because the new “entry . . . in the flow table serves as a fast cache for future lookup of packets belonging to that flow,” and the entry “stores pointers to the appropriate plugins.” *Id.* at 5. Performance is thus enhanced for subsequent packets of the flow, since “[u]sually, filter table lookups are much slower than flow table lookups.” *Id.* *See also id.* at 3 (“Subsequent packets get this information from a fast flow cache which temporarily stores the information gathered by processing the first packet.”).

Decasper98 assigns the sequence of plugins to the flow on the basis of lookups in multiple independent “filter tables.” *E.g., id.* at 5-7 (“The processing of the first packet of a new flow . . . involves n filter table lookup to create a single entry in the flow table for the new

flow”), 7 (“multiple lookups (in different filter tables)”). *E.g.*, a first filter table determines whether a first plugin is added to the sequence, a second independent filter table determines whether a second plugin is added, a third independent filter table determines whether a third plugin is added, and so on. *See id.* at 5-7.

This leads “exponentially” to an enormous number of possible sequences that might be applied to the first packet of a flow when it arrives, “even with very few installed filters.” *See id.* at 7.²¹ These various possible sequences are not stored or enumerated anywhere in the system ahead of time. Instead, the sequence of plugins for a flow is generated algorithmically when the first packet of a flow arrives, by applying a series of filter operation to packet data which was not available to the system until that moment. *See id.* at 5-7.

Decasper98 explicitly considers and rejects a “theoretically possible” alternative approach, which is to replace this system of multiple independent filters with “a single global filter table.” *Id.* at 7. Under this alternative approach, only a single filter would apply to a particular flow, and that single filter would specify the entire sequence of components to be applied to it. *See id.* When the first packet arrived, the system would find the single matching filter and then essentially just read off the sequence of components to be applied to that flow. *See id.* Thus, the sequence would be pre-defined and readily identifiable as such in a specific filter entry, even before the first packet arrived.

However, Decasper98 rejects this approach as “practically infeasible because the space requirements for the global table can, even with very few installed filters, increase very quickly (exponentially) to unacceptable levels.” *Id.* In other words, Decasper98’s multiple filter table

²¹ Compare to Implicit’s Infringement Contentions for this same claim element: “Because of the configurability of policy expressions, and traffic/applications specifications, there are near infinite resultant processing sequences – non-predefined – which will execute.” Ex. 36-B at 16-17.

approach implies so many potential valid sequences that it is impossible to even enumerate them all ahead of time in memory—since they would not fit.

Instead, Decasper98 adopts an algorithmic approach where the correct sequence is generated *dynamically* on demand, by applying the series of multiple filters to the first packet when it arrives. Thus, under Implicit’s apparent claim constructions, Decasper98 discloses “for the first packet of the message, dynamically identifying a non-predefined sequence of components for processing the packets of the message.”

Decasper98 discloses this element in another manner as well. Implicit has characterized the “dynamically identifying” element as encompassing the ability of a network “administrator” to modify or create “Policy Files to change how traffic is managed at runtime.” Ex. 37-D [Implicit Technical Tutorial] at 35; *see generally id.* at 26-42. For example, Implicit has applied this claim construction to the example of a “system administrator” who can “dynamically” implement changing policies to block or permit access to YouTube for certain times or users:

The beauty – and object – of the Implicit system lay in its flexibility. Since a stateful path was not identified and instantiated until *post-first packet*, the system could be changed, *dynamically* on the fly. New components could be added, new rules or policies developed, all as new needs arose. For example, a system administrator could decide how to process particular types of traffic (no You Tube between noon and one) and then *change the rules – or policies* – the next minute or the next day (only CEO gets You Tube).

Ex. 37-A [Implicit Opening Claim Construction Brief] at 5 (emphasis added).

Decasper98 discloses “dynamically identifying” under this apparent claim construction as well:

Shown below are the commands necessary to load and configure [a particular packet scheduling] plugin; this will give the reader a feel for **the simplicity and elegance** with which plugins can be put into operation. Note that these commands *can be executed at any time, even when network traffic is transiting through the system*

Loading the plugin [specific load command given]

Creating an instance [specific create command given]

Registering an instance [specific registration command given]

Adding a filter: this specifies a filter which matches all traffic originating at IPv6 source address 3ffe:2000:400:11::4 and sets the reserved bandwidth for all flows matching this filter to 80%

[specific filter command given]

From now on, all flows originating from the specified source address will get at least 80% of the link bandwidth. Note that [this plugin] *can be turned off any time* by unbinding or freeing the instance or unloading the plugin module (which frees all instances of the plugin automatically).

Ex. 25 at 9-10 (emphasis added). Because an administrator can add and configure plugins “at any time, even when network traffic is transiting through the system,” Decasper98 clearly reads on this Implicit construction of these “dynamically identifying” claim elements as well. *Id.* at 9.

Thus in at least two manners, Decasper98 discloses this “dynamically identifying” claim element under Implicit’s apparent claim constructions.

Decasper98 also teaches “that the output format of the components of the non-predefined sequence match the input format of the next component in the non-predefined sequence.” *See* Claim 1(ii) (showing “providing a plurality of components, each component being a software routine for converting data with an input format into data with an output format”) above.

Decasper98 also teaches “selecting individual components to create the non-predefined sequence of components after the first packet is received.” As explained above, after the first packet of a flow arrives, Descasper98 applies a series of independent filters to it, each of which may select a different individual plugin. *Id.* at 5-7. *See also, e.g., id.* at 4 (Figure 2, showing various individual plugins that might be selected within each category, *e.g.*, “BMP1 BMP2 BMP3”). The very purpose of this architecture is to apply the right specific individual plugins in

a tailored manner to each particular flow. *E.g., id.* at 2 (“it is very important to be able to quickly and efficiently classify packets into flows, and to apply different policies to different flows”), 3, 7.

iv. “storing an indication of . . . the identified components”

Claim 1 further recites “storing an indication of each of the identified components so that the non-predefined sequence does not need to be re-identified for subsequent packets of the message.” Under Implicit’s apparent claim constructions, Decasper98 discloses this element.

Decasper98 explains each flow has its own “entry” in a “flow table.” *Id.* at 5 (including “a flow table entry unambiguously identifies a particular flow”). As the sequence of individual plugins for a flow is determined by applying a series of “filter tables,” pointers to the correct plugin instances are returned and then “stored in the row of the flow table which corresponds to our packet’s flow.” *See id.* at 5-6. And thus, “[e]ach flow table entry stores pointers to the appropriate plugins.” *Id.* at 5.

Decasper98 further explains the reason for storing an indication of each identified plugin component in this manner is to enhance performance: “filter table lookups are much slower than flow table lookups,” and the “entry for a flow in the flow table serves as a fast cache for future lookups of packets belonging to that flow.” *Id.* at 5. Thus, the initial cycle of evaluating filter tables “is executed only for the first packet arriving on an uncached flow,” while “[s]ubsequent packets follow a faster path because of the cached entry in the flow table.” *Id.* at 5-6.

v. “state information”

Claim 1 further recites “for each of a plurality of packets of the message in sequence, for each of a plurality of components in the identified non-predefined sequence, **retrieving** state information relating to performing the processing of the component with the previous packet of the message; **performing** the processing of the identified component with the packet and the

retrieved state information; and **storing** state information relating to the processing of the component with the packet for use when processing the next packet of the message.” Under Implicit’s apparent claim constructions, Decasper98 discloses this “state information” element.

Implicit has taken a broad view of the “state information” limitations, arguing that they cover the retrieval, use, and storage of the identified sequence of components (*e.g.*, a flow record) after the first packet is received. *See* Section IV.C. As demonstrated above (for the “storing an indication” element), Decasper98 retrieves, uses, and stores flow records in this manner to facilitate processing of packets in the same message after the first packet is received and a flow entry built.

Decasper98 also discloses the retrieval, use, and storage of state information on a component-by-component basis, as shown by the following examples. Since Decasper98 teaches that its plugin components are selected on the basis of separate, independent filter tables, virtually any combination of these components could be applied to a particular flow.

(a) IP security components

As a first group of examples, Decasper98 teaches multiple plugin components for implementing “IP security,” which refers to IP security standards including RFC 1825 (“Security Architecture for the Internet Protocol”). *See* Ex. 25 at 2 (“plugins for IP Security” citing footnote “[2]”), 12 (footnote “[2]” citing “RFC 1825”). RFC 1825 explains that various forms of state information would be maintained by these components, including, *e.g.*, “Key(s) used with the authentication algorithm”; “Key(s) used with the encryption algorithm”; “Authentication algorithm and algorithm mode being used”; “Encryption algorithm, algorithm mode, and transform being used”; “cryptographic synchronisation or initialisation vector field for the encryption algorithm”; “Lifetime of the key or time when key change should occur”; and

“Lifetime of [the] Security Association.” Ex. 9 (RFC 1825) at 5-6.²² Under Implicit’s apparent claim constructions, maintaining such state information would read on these “state information” claim elements.

(b) IPv6 options components

As a second group of examples, Decasper98 teaches multiple “plugins implementing IPv6 options.” Ex. 25 at 4. Decasper98 cites the IPv6 specification (RFC 1883), which explains how state information may be maintained on a per-flow basis to support IPv6 options. *See id.* (Decasper98) at 12 (citation to “RFC 1883”: “Internet Protocol, Version 6 (IPv6) Specification”); Ex. 28 (RFC 1883) at 29 (“Routers are free to . . . set up flow-handling state for any flow . . . a router may process its IPv6 header . . . includ[ing] . . . updating a hop-by-hop option The router may then choose to ‘remember’ the results of those processing steps and cache that information Subsequent packets . . . may then be handled by referring to the cached information”). Under Implicit’s apparent claim constructions, maintaining such state information would read on this “state information” element.

(c) statistics gathering component

As another example, Decasper98 teaches “a statistics gathering plugin for network management applications.” Ex. 25 at 4. “[N]etwork management applications . . . typically need to monitor transit traffic at routers in the network, and to gather and report various statistics thereof.” *Id.* at 2. In order to gather such statistics, this plugin would clearly need to maintain state information: *e.g.*, arithmetic counts of bytes or packets through the router which would be retrieved, updated, and stored again with each packet. Under Implicit’s apparent claim

²² As explained above, RFC 1825 is relied on by Decasper98, and is cited throughout this section solely to help explain Decasper98. *See* MPEP § 2205; Ex. 25 at 2, 7, 12 (Decasper98 citations to RFC 1825).

constructions, maintaining such state information would read on this “state information” element.

(d) packet scheduling component

As another example, Decasper98 teaches “packet scheduling plugins,” including one called “Deficient Round Robin [DRR]” which “implement[s] fair queuing among . . . flows.” *Id.* at 9. Using DRR, it is possible for an administrator to stipulate that “all flows matching” a certain “filter” should “get at least 80% of the link bandwidth.” *Id.* at 10. In order for the plugin to enforce this limit, it would need to keep running track of the amount of bandwidth these flows are using—which would entail updating state information regarding bandwidth consumption as the packets in these flows are processed (and thereby contribute to that consumption). Under Implicit’s apparent claim constructions, maintaining such state information would read on this “state information” element.

(b) Claim 15

i. “demultiplexing packets of messages”

Claim 15 recites: “A method in a computer system demultiplexing packets of messages.” Decasper98 discloses these elements. *See* Claim 1(i) (showing “A method in a computer system for processing a message having a sequence of packets”) above. Under Implicit’s apparent claim constructions, “demultiplexing” a packet is satisfied by routing a packet to the correct sequence of components for processing it—and Decasper98 performs this function. *See* Section IV.C and, *e.g.*, Claim 1(iii) (showing “dynamically identifying a non-predefined sequence of components for processing the packets of the message”) above.

ii. “dynamically identifying a non-predefined sequence”

Claim 15 further recites “dynamically identifying a non-predefined sequence of components for processing each message based on the first packet of the message so that

subsequent packets of the message can be processed without re-identifying the components.” Under Implicit’s apparent claim constructions, Decasper98 discloses this element. *See* Claim 1(iii) (showing “for the first packet of the message, dynamically identifying a non-predefined sequence of components for processing the packets of the message”) and Claim 1(iv) (showing “storing an indication of each of the identified components so that the non-predefined sequence does not need to be re-identified for subsequent packets of the message”) above.

iii. “different . . . sequences of components can be identified”

Claim 15 further recites “different non-predefined sequences of components can be identified for different messages.” Under Implicit’s apparent claim constructions, Decasper98 discloses this element.

Decasper98 explains: “it is very important to be able to . . . apply different policies to different flows.” Ex. 25 at 2. This is why Decasper98 applies a series of filters to each flow, wherein each filter may select a specific plugin component implementing a different policy. *See id.* at 5-7.

iv. “each component being a software routine”

Claim 15 further recites “each component being a software routine.” Under Implicit’s apparent claim constructions, Decasper98 discloses this element. *See* Claim 1(ii) (showing “each component being a software routine”) above.

v. “selecting individual components”

Claim 15 further recites “dynamically identifying includes selecting individual components to create the non-predefined sequence of components.” Under Implicit’s apparent claim constructions, Decasper98 discloses this element. *See* Claim 1(iii) (showing “dynamically identifying includes selecting individual components to create the non-predefined sequence of components”) above.

vi. *“state information”*

Claim 15 finally recites “for each packet of each message, performing the processing of the identified non-predefined sequence of components of the message wherein state information generated by performing the processing of a component for a packet is available to the component when the component processes the next packet of the message.” Under Implicit’s apparent claim constructions, Decasper98 discloses this element. *See* Claim 1(v) (showing similar “state information” element) above..

(c) *Claim 35*

i. *“instructions for demultiplexing packets of messages”*

Claim 35 recites: “A computer-readable medium containing instructions for demultiplexing packets of messages.” Under Implicit’s apparent claim constructions, Decasper98 discloses this element.

Decasper98 teaches “router plugins” which are “software modules that are dynamically loaded into the kernel and are responsible for performing certain specific functions on specified network flows.” Ex. 25 at 2. One of ordinary skill would recognize such software modules would be dynamically loaded from a “computer-readable medium,” such as a hard disk in the device. *See also* Claim 15(i) showing (“demultiplexing packets of messages”) above.

ii. *“dynamically identifying a . . . non-predefined sequence”*

Claim 35 recites: “dynamically identifying a message-specific non-predefined sequence of components for processing the packets of each message upon receiving the first packet of the message.” Under Implicit’s apparent claim constructions, Decasper98 discloses this element. *See* Claim 1(iii) (showing “for the first packet of the message, dynamically identifying a non-predefined sequence of components for processing the packets of the message”) above.

iii. *“subsequent packets . . . can use the . . . sequence”*

Claim 35 further recites that “subsequent packets of the message can use the message-specific non-predefined sequence identified when the first packet was received.” Under Implicit’s apparent claim constructions, Decasper98 discloses this element. *See* Claim 1(iv) (showing “storing an indication of each of the identified components so that the non-predefined sequence does not need to be re-identified for subsequent packets of the message”) above.

iv. “selecting individual components”

Claim 35 further recites that “dynamically identifying includes selecting individual components to create the message-specific non-predefined sequence of components.” Under Implicit’s apparent claim constructions, Decasper98 discloses this element. *See* Claim 1(iii) (showing “dynamically identifying includes selecting individual components to create the non-predefined sequence of components after the first packet is received”) above.

v. “state information”

Claim 35 further recites “for each packet of the message, invoking the identified non-predefined sequence of components in sequence to perform the processing of each component for the packet wherein each component saves message-specific state information so that that component can use the save message-specific state information when that component performs its processing on the next packet of the message.” Under Implicit’s apparent claim constructions, Decasper98 discloses this element. *See* Claim 1(v) (showing similar “state information” element) above.

2. Decasper98 Renders Obvious Claims 1, 15, and 35 Under § 102(a), (b)

If certain aspects recited in claims 1, 15, and 35 of the ‘163 patent are not deemed to be disclosed or inherent over Decasper98, then the inclusion of those aspects certainly would be obvious over Decasper98, under 35 U.S.C. § 103.

(a) Claim 1

i. “A method in a computer system”

Claim 1 recites: “A method in a computer system for processing a message having a sequence of packets, the method comprising” Under Implicit’s apparent claim constructions, Decasper98 renders obvious this element.

Decasper98 teaches “an extensible and modular software architecture for high-performance . . . routers” which “allows code modules called plugins to be dynamically loaded into the kernel and configured at run time.” Ex. 25 at 11. It was obvious to run this software architecture on a “computer system.”

Claim 1 further recites the method is “for processing a message having a sequence of packets.” Under Implicit’s apparent claim constructions, Decasper98 discloses this element. *See* Section V.C.1 (Decasper98 102) at Claim 1(i) (showing same element).

ii. “a plurality of components”

Claim 1 further recites: “providing a plurality of components, each component being a software routine for converting data with an input format into data with an output format.” Under Implicit’s apparent claim constructions, Decasper98 renders obvious this element.

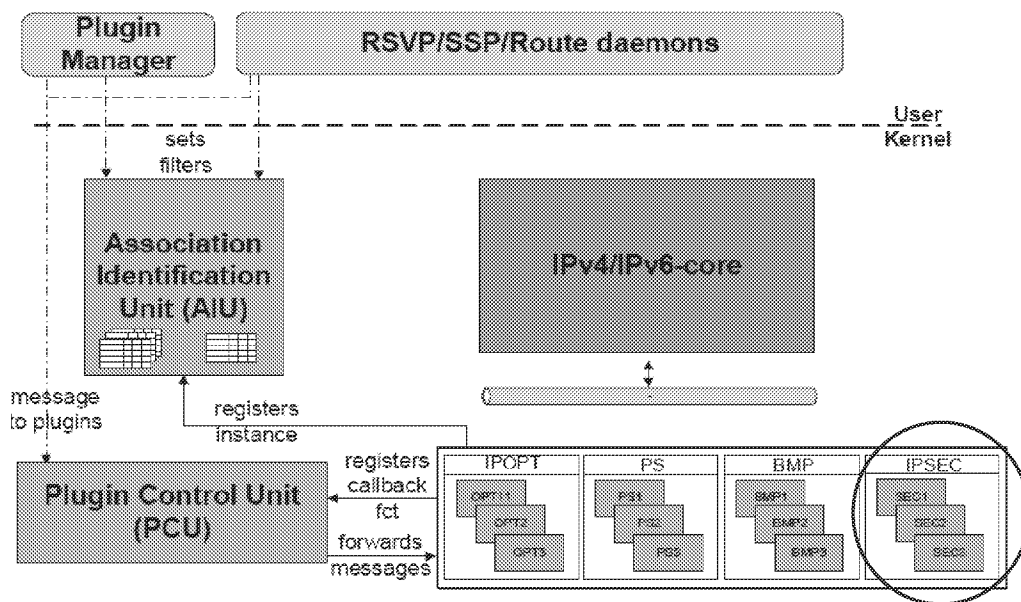
Decasper98 teaches: “One of the novel features of our design is the ability to bind different plugins to individual flows.” Ex. 25 at 1. “[P]lugins are . . . software modules that are . . . responsible for performing certain functions on specified network flows,” and to the extent a software module differs from a software routine, it was obvious to implement these plugins as software routines.

These “plugins” also convert data with an input format into data with an output format. Under Implicit’s apparent claim construction, obvious implementations of all of the plugins taught by Decasper98 would read on these claim elements, including “plugins implementing IPv6 options, plugins for packet scheduling . . . plugins for IP security a routing plugin, a

statistics gathering plugin . . . a firewall plugin,” and so on. Ex. 25 at 4. Additional examples are set forth below.

(a) IP security components

As a first group of examples, Decasper98 teaches multiple plugin components for implementing “IP security,” which refers to IP security standards including RFC 1825 (“Security Architecture for the Internet Protocol”), RFC 1826 (“IP Authentication Header”), and RFC 1827 (“IP Encapsulating Security Payload”). See Ex. 25 at 2 (“plugins for IP Security” citing footnote “[2]”), 12 (footnote “[2]” citing “RFC 1825”); Ex. 26 (RFC 1825) at 3 (citing “two specific headers . . . used to provide security in IPv4 and IPv6” which may be added to a packet: an authentication header (detailed in RFC 1826) and an encryption header (detailed in RFC 1827)).²³



Ex. 25 at 4 (Figure 2, showing multiple “IPSEC” plugins: “SEC1 SEC2 SEC3”).

²³ RFC 1825 is relied on by Decasper98, and is cited throughout this section solely to help explain Decasper98. See MPEP § 2205; Ex. 25 at 2, 7, 12 (Decasper98 citations to RFC 1825).

Since there are multiple distinct and independent IP security operations which may be performed (*e.g.*, encryption and/or authentication) and multiple distinct IP security plugins (*e.g.*, “SEC1 SEC2 SEC3”), it was obvious to at least provide a distinct plugin for authentication and a second distinct plugin for encryption.

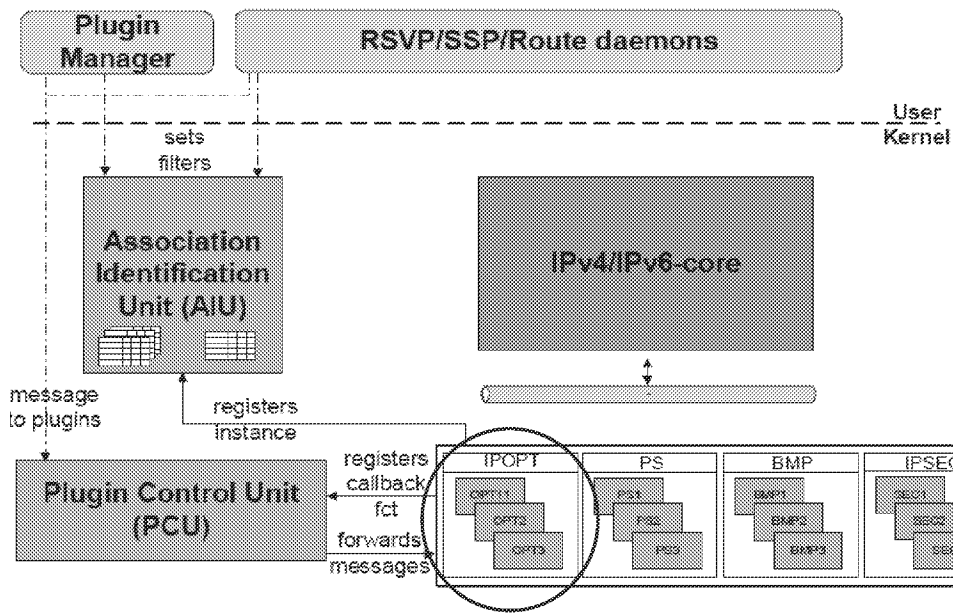
A plugin component which performed encryption on a packet in the manner described by RFC 1825 would add an “IP Encapsulating Security Payload . . . header” to the packet. *See* Ex. 26 (RFC 1825) at 3. Under Implicit’s apparent claim constructions, this would comprise “converting data with an input format into data with an output format.”

A plugin component which performed authentication on a packet in the manner described by RFC 1825 would add an “IP Authentication Header” to the packet. *Id.* at 3. Under Implicit’s apparent claim constructions, this would comprise “converting data with an input format into data with an output format.”

Thus, considering merely these two obvious plugin components alone, they would comprise “a plurality of components, each component being a software routine for converting data with an input format into data with an output format,” under Implicit’s apparent claim constructions.

(b) IPv6 options components

As a second group of examples, Decasper98 teaches multiple “plugins implementing IPv6 options.” Ex. 25 at 4.



Id. at 4 (Figure 2, showing multiple “IPOPT” plugins: “OPT1 OPT2 OPT3”).

Given the known “onion-peeling procedure” which is used to process these IPv6 option headers, it was at least obvious for these IPv6 options components to add or remove these headers in the course of processing a packet.²⁴ Under Implicit’s apparent claim constructions, this would comprise “converting data with an input format into data with an output format.”

iii. “dynamically identifying a non-predefined sequence”

Claim 1 further recites: “for the first packet of the message, dynamically identifying a non-predefined sequence of components for processing the packets of the message such that the output format of the components of the non-predefined sequence match the input format of the next component in the non-predefined sequence, wherein dynamically identifying includes selecting individual components to create the non-predefined sequence of components after the

²⁴ See, e.g., Ex. 29 (“IPv6: The New Internet Protocol” by Christian Huitema) (1997) at 15 (figure showing “Daisy Chain” of IPv6 “extension headers”), 25 (“recommended order” of IPv6 extension headers includes “2. Hop-by-Hop options header 3. Destination options header”; “‘onion-peeling’ procedure” for extension headers wherein “[e]ach successive layer would be processed in turn, just like removing each layer of an onion in turn”). This reference is cited in this context solely to help explain Decasper98, which refers to IPv6 options. See MPEP § 2205.

first packet is received.” Under Implicit’s apparent claim constructions, Decasper98 renders obvious this element.

As explained above, Decasper98 discloses this element in at least two distinct manners. See Section V.C.1 (Decasper98 102) at Claim 1(iii).

Regarding the limitation “such that the output format of the components . . . match the input format of the next component,” it was well-known to those of ordinary skill in the art that certain operations on a packet must be performed in a certain order: *e.g.*, if a packet is first converted *in toto* into an encrypted format by a first component, a subsequent component would be unable to, *e.g.*, process any IPv6 option headers in the packet, or to insert any new ones (because it was expecting to receive the packet in an unencrypted format). Thus, it was certainly obvious for one of ordinary skill in the art to arrange the sequence of components in a compatible manner, such that the output format of one matches the input format of the next.

iv. “storing an indication of . . . the identified components”

Claim 1 further recites “storing an indication of each of the identified components so that the non-predefined sequence does not need to be re-identified for subsequent packets of the message.” Under Implicit’s apparent claim constructions, Decasper98 discloses this element. See Section V.C.1 (Decasper98 102) at Claim 1(iv).

v. “state information”

Claim 1 further recites “for each of a plurality of packets of the message in sequence, for each of a plurality of components in the identified non-predefined sequence, **retrieving** state information relating to performing the processing of the component with the previous packet of the message; **performing** the processing of the identified component with the packet and the retrieved state information; and **storing** state information relating to the processing of the component with the packet for use when processing the next packet of the message.” Under

Implicit's apparent claim constructions, Decasper98 renders obvious this "state information" element.

Implicit has taken a broad view of the "state information" limitations, arguing that they cover the retrieval, use, and storage of the identified sequence of components (*e.g.*, a flow record) after the first packet is received. *See* Section IV.C. As demonstrated above (for the "storing an indication" element), Decasper98 retrieves, uses, and stores flow records in this manner to facilitate processing of packets in the same message after the first packet is received and a flow entry built.

Decasper98 also renders obvious the retrieval, use, and storage of state information on a component-by-component basis, as shown by the following examples. Since Decasper98 teaches that its plugin components are selected on the basis of separate, independent filter tables, it was obvious that virtually any combination of these components would be applied to a particular flow.

(a) IP security components

As a first group of examples, Decasper98 teaches multiple plugin components for implementing "IP security," which refers to IP security standards including RFC 1825 ("Security Architecture for the Internet Protocol"), RFC 1826 ("IP Authentication Header") (describing authentication), RFC 1827 ("IP Encapsulating Security Payload") (describing encryption), and RFC 1829 ("The ESP DES-CBC Transform") (describing an algorithm which "MUST" be supported for encrypting packets). *See* Ex. 25 (Decasper98) at 2 ("plugins for IP Security" citing

footnote “[2]”), 12 (footnote “[2]” citing “RFC 1825”); Ex. 26 (RFC 1825) at 10 (“MUST support”), 19-21 (citing “RFC 1826,” “RFC 1827,” “RFC 1829”).²⁵

As explained above, it was obvious to provide separate plugin components for encryption and authentication. *See* Claim 1(ii)(a) above.

RFC 1825 explains that various forms of state information would be maintained by these components, including, *e.g.*, “Key(s) used with the authentication algorithm”; “Key(s) used with the encryption algorithm”; “Authentication algorithm and algorithm mode being used”; “Encryption algorithm, algorithm mode, and transform being used”; “cryptographic synchronisation or initialisation vector field for the encryption algorithm”; “Lifetime of the key or time when key change should occur”; and “Lifetime of [the] Security Association.” Ex. 9 (RFC 1825) at 5-6. Obvious implementations to maintain this state information would read on this claim element, under Implicit’s apparent claim constructions. For example, both the encryption and authentication component would maintain “Lifetime of the key or time when key change should occur.” *See id.* Maintaining a “Lifetime of the key” (as opposed to maintaining “time when key change should occur”) at least renders obvious a countdown implementation wherein the remaining lifetime is updated with each invocation of the component.

Additionally, regarding the encryption component in particular, an obvious implementation of its encryption technique would read on this claim element in still another manner, under Implicit’s apparent claim constructions. RFC 1825 explains that the encryption algorithm of RFC 1829 “MUST” be supported for encrypting packets. Ex. 26 (RFC 1825) at 10 (“the IP Encapsulating Security Payload MUST support the use of the Data Encryption Standard

²⁵ As explained above, RFC 1825 is relied on by Decasper98, and it and another standard it cites (RFC 1829) are cited throughout this section solely to help explain Decasper98. *See* MPEP § 2205; Ex. 25 at 2, 7, 12 (Decasper98 citations to RFC 1825).

(DES) in Cipher-Block Chaining (CBC) Mode”), 21 (citing RFC 1829: “The ESP DES-CBC Transform”). RFC 1829 explains that in order to apply its encryption technique, “an Initialization Vector (IV) that is eight octets in length” must be placed in “[e]ach datagram” to be encrypted (*i.e.*, in each packet). *See* Ex. 27 (which is RFC 1829: “The ESP DES-CBC Transform” by P. Karn et al.) (1995) at 1. RFC 1829 further explains that while the “method for selection of IV values is implementation dependent,” a “common acceptable technique is simply a counter, beginning with a random chosen value.” *Id.* One of ordinary skill would therefore be familiar with this counter technique, and find it obvious to apply to the encryption component of Decasper98. Doing so would clearly entail, for each packet: *e.g.*, retrieving the previous counter value, applying it the packet it, incrementing the counter value, and storing it for use when encrypting the next packet. Under Implicit’s apparent claim constructions, this obvious implementation would read on this claim element.

(b) IPv6 options components

As a second group of examples, Decasper98 teaches multiple “plugins implementing IPv6 options.” Ex. 25 at 4. Decasper98 cites the IPv6 specification (RFC 1883), which explains how state information may be maintained on a per-flow basis to support IPv6 options. *See id.* (Decasper98) at 12 (citation to “RFC 1883”: “Internet Protocol, Version 6 (IPv6) Specification”); Ex. 28 (RFC 1883) at 29 (“Routers are free to . . . set up flow-handling state for any flow . . . a router may process its IPv6 header . . . includ[ing] . . . updating a hop-by-hop option . . . The router may then choose to ‘remember’ the results of those processing steps and cache that information . . . Subsequent packets . . . may then be handled by referring to the cached information”). Because Decasper98 teaches “flows” and RFC 1883 explicitly discloses this flow-based technique for processing IPv6 options, it was certainly at least obvious that

Decasper98's IPv6 options plugins would apply the technique. Under Implicit's apparent claim constructions, such maintenance of state information would read on this "state information" element.

(c) statistics gathering component

As another example, Decasper98 teaches "a statistics gathering plugin for network management applications." Ex. 25 at 4. "[N]etwork management applications . . . typically need to monitor transit traffic at routers in the network, and to gather and report various statistics thereof." *Id.* at 2. In order to gather such statistics, it was certainly at least obvious for this plugin to maintain state information comprising, *e.g.*, arithmetic counts of bytes or packets through the router which would be retrieved, updated, and stored again with each packet. Under Implicit's apparent claim constructions, such maintenance of state information would read on this "state information" element.

(d) packet scheduling component

As another example, Decasper98 teaches "two packet scheduling plugins," including one called "Deficient Round Robin [DRR]" which "provides fair link bandwidth distribution among different flows." *Id.* at 9. In order for the plugin component to enforce such bandwidth distribution, it was certainly at least obvious for it to maintain state information tracking the amount of bandwidth being used, and to update this state information upon processing each packet to reflect the packet's contribution to this bandwidth. Under Implicit's apparent claim constructions, such maintenance of state information would read on this "state information" element.

(e) firewall component

As another example, Decasper98 teaches “a firewall plugin.” *Id.* at 4. It was well-known to those of ordinary skill in the art that it was useful for firewall functions to be implemented in a stateful manner, such that previously seen packets affect the processing of subsequent packets.²⁶ Under Implicit’s apparent claim constructions, such maintenance of state information of state information would read on this “state information” element.

(b) *Claim 15*

i. *“demultiplexing packets of messages”*

Claim 15 recites: “A method in a computer system demultiplexing packets of messages.” Under Implicit’s apparent claim constructions, Decasper98 discloses “demultiplexing packets of messages” and renders obvious that the method would be performed in a “computer system.” *See* Claim 1(i) and Section V.C.1 (Decasper98 102) at Claim 15(i) above.

ii. *“dynamically identifying a non-predefined sequence”*

Claim 15 further recites “dynamically identifying a non-predefined sequence of components for processing each message based on the first packet of the message so that subsequent packets of the message can be processed without re-identifying the components.” Under Implicit’s apparent claim constructions, Decasper98 discloses this element. *See* Section V.C.1 (Decasper98 102) at Claim 1(iii) (showing “for the first packet of the message, dynamically identifying a non-predefined sequence of components for processing the packets of the message”) and Claim 1(iv) (showing “storing an indication of each of the identified components so that the non-predefined sequence does not need to be re-identified for subsequent packets of the message”) above.

iii. *“different . . . sequences of components can be identified”*

²⁶ See, for example, the Shwed and Checkpoint references cited herein.

Claim 15 further recites “different non-predefined sequences of components can be identified for different messages.” Under Implicit’s apparent claim constructions, Decasper98 discloses this element. *See* Section V.C.1 (Decasper98 102) at Claim 15(iii) above.

iv. “each component being a software routine”

Claim 15 further recites “each component being a software routine.” Under Implicit’s apparent claim constructions, Decasper98 renders obvious this element. *See* Claim 1(ii) (showing same element) above.

v. “selecting individual components”

Claim 15 further recites “dynamically identifying includes selecting individual components to create the non-predefined sequence of components.” Under Implicit’s apparent claim constructions, Decasper98 discloses this element. *See* Section V.C.1 (Decasper98 102) at Claim 1(iii) (showing same element) above.

vi. “state information”

Claim 15 finally recites “for each packet of each message, performing the processing of the identified non-predefined sequence of components of the message wherein state information generated by performing the processing of a component for a packet is available to the component when the component processes the next packet of the message.” Under Implicit’s apparent claim constructions, Decasper98 renders obvious this element. *See* Claim 1(v) above (showing similar “state information” element).

(c) Claim 35

i. “instructions for demultiplexing packets of messages”

Claim 35 recites: “A computer-readable medium containing instructions for demultiplexing packets of messages.” Under Implicit’s apparent claim constructions, Decasper98 renders obvious this element.

Decasper98 teaches “router plugins” which are “software modules that are dynamically loaded into the kernel and are responsible for performing certain specific functions on specified network flows.” Ex. 25 at 2. It was obvious that such software modules would be dynamically loaded from a “computer-readable medium,” such as a hard disk in the device.

ii. “dynamically identifying a . . . non-predefined sequence”

Claim 35 recites: “dynamically identifying a message-specific non-predefined sequence of components for processing the packets of each message upon receiving the first packet of the message.” Under Implicit’s apparent claim constructions, Decasper98 discloses this element. *See* Section V.C.1 (Decasper98 102) at Claim 1(iii) (showing “for the first packet of the message, dynamically identifying a non-predefined sequence of components for processing the packets of the message”) and Claim 15(iii) (showing “different non-predefined sequences of components can be identified for different messages”) above.

iii. “subsequent packets . . . can use the . . . sequence”

Claim 35 further recites that “subsequent packets of the message can use the message-specific non-predefined sequence identified when the first packet was received.” Under Implicit’s apparent claim constructions, Decasper98 discloses this element. *See* Section V.C.1 (Decasper98 102) at Claim 1(iv) (showing “storing an indication of each of the identified components so that the non-predefined sequence does not need to be re-identified for subsequent packets of the message”) above.

iv. “selecting individual components”

Claim 35 further recites that “dynamically identifying includes selecting individual components to create the message-specific non-predefined sequence of components.” Under Implicit’s apparent claim constructions, Decasper98 discloses this element. *See* Section V.C.1

(Decasper98 102) at Claim 1(iii) (showing “dynamically identifying includes selecting individual components to create the non-predefined sequence of components”) above.

v. “state information”

Claim 35 further recites “for each packet of the message, invoking the identified non-predefined sequence of components in sequence to perform the processing of each component for the packet wherein each component saves message-specific state information so that that component can use the save message-specific state information when that component performs its processing on the next packet of the message.” Under Implicit’s apparent claim constructions, Decasper98 renders obvious this element. *See* Claim 1(v) above (showing similar “state information” element).

3. Decasper98 in View of RFC 1825 and RFC 1829 Renders Obvious Claims 1, 15, and 35 Under § 103

The specification RFC 1825 (“Security Architecture for the Internet Protocol”) (Ex. 26, “RFC 1825”) by R. Atkinson was published in August 1995. The specification RFC 1829 (“The ESP DES-CBC Transform”) by P. Karn *et al.* was also published in August 1995. Neither was considered during the prosecution of the ‘163 patent.

If certain aspects recited in claims 1, 15, and 35 of the ‘163 patent are not deemed to be disclosed, inherent, or obvious over Decasper98 alone, then the inclusion of those aspects certainly would be obvious over Decasper98 in view of RFC 1825 and RFC 1829, under 35 U.S.C. § 103.

It was obvious to supplement the teachings of Decasper98 with RFC 1825 and RFC 1829 because Decasper98 expressly cites RFC 1825 to explain its “plugins for IP Security,” and RFC 1825 expressly cites RFC 1829 to explain an algorithm which “MUST” be supported for encrypting packets. Ex. 25 (Decasper98) at 2 (“plugins for IP Security” citing footnote “[2]”),

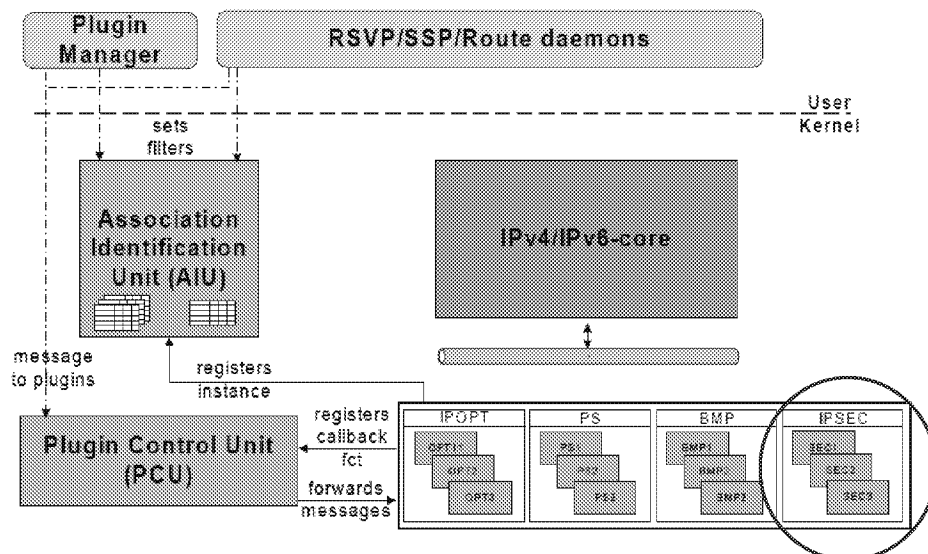
12 (footnote “[2]” citing “RFC 1825”); Ex. 26 (RFC 1825) at 10 (“the IP Encapsulating Security Payload MUST support the use of the Data Encryption Standard (DES) in Cipher-Block Chaining (CBC) Mode”), 21 (citing RFC 1829: “The ESP DES-CBC Transform”).

i. “a plurality of components”

Claim 1 recites in pertinent part: “providing a plurality of components, each component being a software routine for converting data with an input format into data with an output format.” Under Implicit’s apparent claim constructions, Decasper98 in view of RFC 1825 and RFC 1829 renders obvious this element.

(a) IP security components

Decasper98 teaches multiple plugin components for implementing “IP security,” which refers to IP security standards including RFC 1825 (“Security Architecture for the Internet Protocol”), RFC 1826 (“IP Authentication Header”), and RFC 1827 (“IP Encapsulating Security Payload”). See Ex. 25 at 2 (“plugins for IP Security” citing footnote “[2]”), 12 (footnote “[2]” citing “RFC 1825”); Ex. 26 (RFC 1825) at 3 (citing “two specific headers . . . used to provide security in IPv4 and IPv6” which may be added to a packet: an authentication header (detailed in RFC 1826) and an encryption header (detailed in RFC 1827)).



Ex. 25 at 4 (Figure 2, showing multiple “IPSEC” plugins: “SEC1 SEC2 SEC3”).

Since there are multiple distinct and independent IP security operations which may be performed (*e.g.*, encryption and/or authentication) and multiple distinct IP security plugins (*e.g.*, “SEC1 SEC2 SEC3”), it was obvious to at least provide a distinct plugin for authentication and a second distinct plugin for encryption.

A plugin component which performed encryption on a packet in the manner described by RFC 1825 would add an “IP Encapsulating Security Payload . . . header” to the packet. *See* Ex. 26 (RFC 1825) at 3. Under Implicit’s apparent claim constructions, this would comprise “converting data with an input format into data with an output format.”

A plugin component which performed authentication on a packet in the manner described by RFC 1825 would add an “IP Authentication Header” to the packet. *Id.* Under Implicit’s apparent claim constructions, this would comprise “converting data with an input format into data with an output format.”

Thus, considering merely these two obvious plugin components alone, they would comprise “a plurality of components, each component being a software routine for converting data with an input format into data with an output format,” under Implicit’s apparent claim constructions.

ii. “state information”

Claim 1 finally recites in pertinent part: “for each of a plurality of packets of the message in sequence, for each of a plurality of components in the identified non-predefined sequence, **retrieving** state information relating to performing the processing of the component with the previous packet of the message; **performing** the processing of the identified component with the packet and the retrieved state information; and **storing** state information relating to the processing of the component with the packet for use when processing the next packet of the

message.” Under Implicit’s apparent claim constructions, Decasper98 in view of RFC 1825 and RFC 1829 renders obvious this “state information” element.

(a) IP security components

Decasper98 teaches multiple plugin components for implementing “IP security,” which refers to IP security standards including RFC 1825 (“Security Architecture for the Internet Protocol”). *See* Ex. 25 at 2 (“plugins for IP Security” citing footnote “[2]”), 12 (footnote “[2]” citing “RFC 1825”).

As explained above, it was obvious to provide separate plugin components for encryption and authentication. *See* Claim 1(i)(a) above.

RFC 1825 explains that various forms of state information would be maintained by these components, including, *e.g.*, “Key(s) used with the authentication algorithm”; “Key(s) used with the encryption algorithm”; “Authentication algorithm and algorithm mode being used”; “Encryption algorithm, algorithm mode, and transform being used”; “cryptographic synchronisation or initialisation vector field for the encryption algorithm”; “Lifetime of the key or time when key change should occur”; and “Lifetime of [the] Security Association.” Ex. 9 (RFC 1825) at 5-6. Obvious implementations to maintain this state information would read on this claim element, under Implicit’s apparent claim constructions. For example, both the encryption and authentication component would maintain “Lifetime of the key or time when key change should occur.” *See id.* Maintaining a “Lifetime of the key” (as opposed to maintaining “time when key change should occur”) at least renders obvious a countdown implementation wherein the remaining lifetime is updated with each invocation of the component.

Additionally, regarding the encryption component in particular, an obvious implementation of its encryption technique would read on this claim element in still another

manner, under Implicit’s apparent claim constructions. RFC 1825 explains that the encryption algorithm of RFC 1829 “MUST” be supported for encrypting packets. Ex. 26 (RFC 1825) at 10 (“the IP Encapsulating Security Payload MUST support the use of the Data Encryption Standard (DES) in Cipher-Block Chaining (CBC) Mode”), 21 (citing RFC 1829: “The ESP DES-CBC Transform”). RFC 1829 explains that in order to apply its encryption technique, “an Initialization Vector (IV) that is eight octets in length” must be placed in “[e]ach datagram” to be encrypted (*i.e.*, in each packet). Ex. 27 (RFC 1829) at 1. RFC 1829 further explains that while the “method for selection of IV values is implementation dependent,” a “common acceptable technique is simply a counter, beginning with a random chosen value.” *Id.* It was therefore obvious to apply this counter technique to the encryption component of Decasper98. Doing so would clearly entail, for each packet: *e.g.*, retrieving the previous counter value, applying it the packet it, incrementing the counter value, and storing it for use when encrypting the next packet. Under Implicit’s apparent claim constructions, this obvious implementation would read on this claim element.

(b) Claim 15

i. “dynamically identifying a non-predefined sequence”

Claim 15 recites in pertinent part: “dynamically identifying a non-predefined sequence of components for processing each message based on the first packet of the message so that subsequent packets of the message can be processed without re-identifying the components.” Under Implicit’s apparent claim constructions, Decasper98 in view of RFC 1825 and RFC 1829 renders obvious this element.

As explained above, one such obvious sequence would comprise a stateful encryption component and a stateful authentication component. *See* Claim 1 above.

ii. “state information”

Claim 15 finally recites in pertinent part: “for each packet of each message, performing the processing of the identified non-predefined sequence of components of the message wherein state information generated by performing the processing of a component for a packet is available to the component when the component processes the next packet of the message.” Under Implicit’s apparent claim constructions, Decasper98 in view of RFC 1825 and RFC 1829 renders obvious this element. *See* Claim 1(ii) above (showing similar “state information” elements).

(c) Claim 35

i. “dynamically identifying a . . . non-predefined sequence”

Claim 35 recites in pertinent part: “dynamically identifying a message-specific non-predefined sequence of components for processing the packets of each message upon receiving the first packet of the message.” Under Implicit’s apparent claim constructions, Decasper98 in view of RFC 1825 and RFC 1829 renders obvious this element.

As explained above, one such obvious sequence would comprise a stateful encryption component and a stateful authentication component. *See* Claim 1 above.

ii. “state information”

Claim 35 finally recites in pertinent part: “for each packet of the message, invoking the identified non-predefined sequence of components in sequence to perform the processing of each component for the packet wherein each component saves message-specific state information so that that component can use the save message-specific state information when that component performs its processing on the next packet of the message.” Under Implicit’s apparent claim constructions, Decasper98 in view of RFC 1825 and RFC 1829 renders obvious this element. *See* Claim 1(ii) above (showing similar “state information” element).

4. Decasper98 in View of RFC 1883 and Huitema Renders Obvious Claims 1, 15, and 35 Under § 103

The specification RFC 1883 (“Internet Protocol, Version 6 (IPv6) Specification”) (Ex. 28, “RFC 1883”) by S. Deering *et al.* was published in December 1995. The book “IPv6: The New Internet Protocol” (Ex. 29, “Huitema”) by Christian Huitema was published on October 28, 1997. Neither was considered during the prosecution of the ‘163 patent.

If certain aspects recited in claims 1, 15, and 35 of the ‘163 patent are not deemed to be disclosed, inherent, or obvious over Decasper98, then the inclusion of those aspects certainly would be obvious over Decasper98 in view of RFC 1883 and Huitema, under 35 U.S.C. § 103.

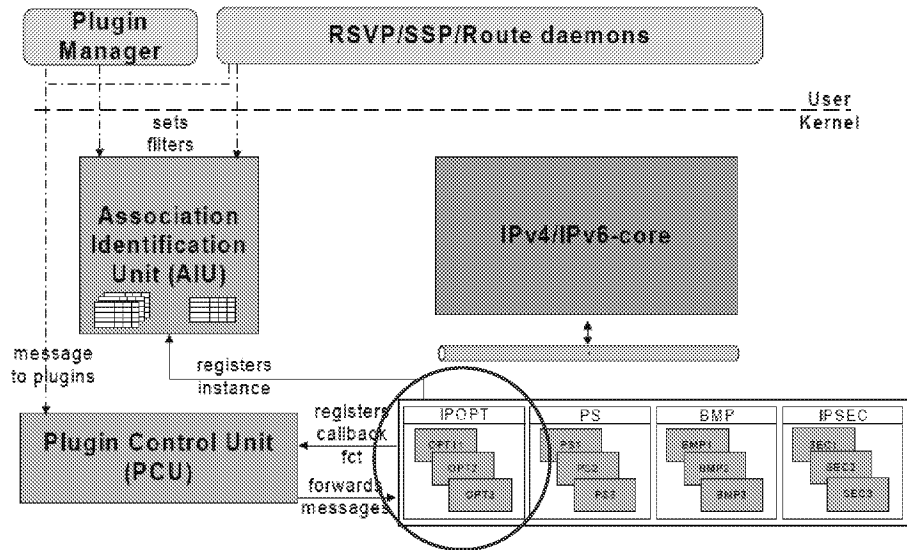
It was obvious to supplement the teachings of Decasper98 with RFC 1883 and Huitema because Decasper98 discloses “plugins implementing IPv6 options,” which are explained by RFC 1883 and Huitema. Ex. 25 at 4 (“plugins implementing IPv6 options”). Moreover, Decasper98 and Huitema expressly cite to RFC 1883. *Id.* at 12 (citation to “RFC 1883”); Ex. 29 at 43.

i. “a plurality of components”

Claim 1 recites in pertinent part: “providing a plurality of components, each component being a software routine for converting data with an input format into data with an output format.” Under Implicit’s apparent claim constructions, Decasper98 in view of RFC 1883 and Huitema renders obvious this element.

(a) IPv6 options components

Decasper98 teaches multiple “plugins implementing IPv6 options.” Ex. 25 at 4.



Id. at 4 (Figure 2, showing multiple “IPOPT” plugins: “OPT1 OPT2 OPT3”).

Given the “onion-peeling procedure” taught by Huitema to process these IPv6 option headers, it was at least obvious for these IPv6 options components to add or remove these headers in the course of processing a packet. *See* Ex. 29 (Huitema) at 15 (figure showing “Daisy Chain” of IPv6 “extension headers”), 25 (“recommended order” of IPv6 extension headers includes “2. Hop-by-Hop options header 3. Destination options header”; “onion-peeling” procedure” for extension headers wherein “[e]ach successive layer would be processed in turn, just like removing each layer of an onion in turn”). Under Implicit’s apparent claim constructions, this would comprise “converting data with an input format into data with an output format.”

Thus, considering merely multiple IPv6 option components alone, or one or more IPv6 option components as combined with other components (*e.g.*, a stateful encryption and/or authentication component), these would comprise “a plurality of components, each component being a software routine for converting data with an input format into data with an output format,” under Implicit’s apparent claim constructions.

ii. “state information”

Claim 1 finally recites in pertinent part: “for each of a plurality of packets of the message in sequence, for each of a plurality of components in the identified non-predefined sequence, **retrieving** state information relating to performing the processing of the component with the previous packet of the message; **performing** the processing of the identified component with the packet and the retrieved state information; and **storing** state information relating to the processing of the component with the packet for use when processing the next packet of the message.” Under Implicit’s apparent claim constructions, Decasper98 in view of RFC 1883 and Huitema renders obvious this “state information” element.

(a) IPv6 options components

Decasper98 teaches multiple “plugins implementing IPv6 options.” Ex. 25 at 4. Decasper98 cites the IPv6 specification (RFC 1883), which explains how state information may be maintained on a per-flow basis to support IPv6 options. *See id.* at 12 (citation to “RFC 1883”); Ex. 28 (RFC 1883) at 29 (“Routers are free to . . . set up flow-handling state for any flow a router may process its IPv6 header includ[ing] . . . updating a hop-by-hop option The router may then choose to ‘remember’ the results of those processing steps and cache that information Subsequent packets . . . may then be handled by referring to the cached information”). Because Decasper98 teaches “flows” and RFC 1883 explicitly discloses this flow-based technique for processing IPv6 options, it was certainly at least obvious for Decasper98’s IPv6 options plugins to apply the technique. Under Implicit’s apparent claim constructions, such maintenance of state information would read on this “state information” element.

(b) Claim 15

i. “dynamically identifying a non-predefined sequence”

Claim 15 recites in pertinent part: “dynamically identifying a non-predefined sequence of components for processing each message based on the first packet of the message so that subsequent packets of the message can be processed without re-identifying the components.” Under Implicit’s apparent claim constructions, Decasper98 in view of RFC 1883 and Huitema renders obvious this element.

As explained above, one such obvious sequence would comprise multiple stateful IPv6 option components, or one or more stateful IPv6 option components as combined with other components (*e.g.*, a stateful encryption and/or stateful authentication component). *See* Claim 1 above.

ii. “state information”

Claim 15 finally recites in pertinent part: “for each packet of each message, performing the processing of the identified non-predefined sequence of components of the message wherein state information generated by performing the processing of a component for a packet is available to the component when the component processes the next packet of the message.” Under Implicit’s apparent claim constructions, Decasper98 in view of RFC 1883 and Huitema renders obvious this element. *See* Claim 1(ii) above (showing similar “state information” elements).

(c) Claim 35

i. “dynamically identifying a . . . non-predefined sequence”

Claim 35 recites in pertinent part: “dynamically identifying a message-specific non-predefined sequence of components for processing the packets of each message upon receiving the first packet of the message.” Under Implicit’s apparent claim constructions, Decasper98 in view of RFC 1883 and Huitema renders obvious this element.

As explained above, one such obvious sequence would comprise multiple stateful IPv6 option components, or one or more stateful IPv6 option components as combined with other components (*e.g.*, a stateful encryption and/or stateful authentication component). *See* Claim 1 above.

ii. “state information”

Claim 35 finally recites in pertinent part: “for each packet of the message, invoking the identified non-predefined sequence of components in sequence to perform the processing of each component for the packet wherein each component saves message-specific state information so that that component can use the save message-specific state information when that component performs its processing on the next packet of the message.” Under Implicit’s apparent claim constructions, Decasper98 in view of RFC 1883 and Huitema renders obvious this element. *See* Claim 1(ii) above (showing similar “state information” element).

5. Decasper98 in View of Decasper97 Renders Obvious Claims 1, 15, and 35 Under § 103

The article “Crossbow: A Toolkit for Integrated Services over Cell Switched IPv6” (Ex. 30, “Decasper97”) by Dan Decasper *et. al* was published on May 29, 1997. Decasper97 was not considered during the prosecution of the ‘163 patent.

If certain aspects recited in claims 1, 15, and 35 of the ‘163 patent are not deemed to be disclosed, inherent, or obvious over Decasper98 alone, then the inclusion of those aspects certainly would be obvious over Decasper98 in view of Decasper97, under 35 U.S.C. § 103.

It was obvious to supplement the teachings of Decasper98 with Decasper97, because both describe a very similar architecture for dynamically loading router components on the basis of independent filters. *Compare* Ex. 25 (Decasper98) at 5 (“entries in the flow table”), 2 (“New plugins can be dynamically loaded at run time”), 5-7 (filter operation), 4 (“plugins implementing

IPv6 options, plugins for packet scheduling . . . and plugins for IP security”); Ex. 30 (Decasper97) at 4 (“Flow entries”), 3 (“dynamically loadable modules”), 3-4 (filter operation), 3-4 (modules include “authentication modules . . . encryption modules . . . IPv6 option modules . . . and packet scheduling modules.”).

(a) Claim 1

Because Decasper98 and Decasper97 are similar in approach and detail, Decasper97 confirms the obviousness of claim 1 in a number of ways. One particularly pertinent aspect of Decasper97 is pointed out below.

i. “a plurality of components”

Claim 1 recites in pertinent part: “providing a plurality of components, each component being a software routine for converting data with an input format into data with an output format.” Under Implicit’s apparent claim constructions, Decasper98 in view of Decasper97 renders obvious this element.

Though it was obvious over Decasper98 alone to employ distinct components for encryption and authentication (since they are distinct operations not always performed together on the same packet), Decasper97 renders this even more obvious by teaching precisely that. *See* Ex. 30 at 3 (“Five different module types are supported in the initial version,” including “authentication modules” and “encryption modules”).

One of ordinary skill was aware that encryption and authentication each typically operate by inserting an additional header in the packet. Adding such a header would comprise “converting data with an input format into data with an output format” under Implicit’s apparent claim constructions.

Thus, considering merely an encryption component and an authentication component assigned to a particular flow, these alone would comprise “a plurality of components” which read on this claim elements.

Since Decasper98 teaches that its plugins components are selected on the basis of separate, independent filter tables, it was obvious that virtually any combination of the components discussed below would be applied to a particular flow, including (as here) the combination of an encryption and an authentication component. Because encryption and authentication operations often go hand in hand, the combination of an encryption and an authentication component was a particularly obvious one of these many combinations.

ii. “state information”

Claim 1 finally recites in pertinent part: “for each of a plurality of packets of the message in sequence, for each of a plurality of components in the identified non-predefined sequence, **retrieving** state information relating to performing the processing of the component with the previous packet of the message; **performing** the processing of the identified component with the packet and the retrieved state information; and **storing** state information relating to the processing of the component with the packet for use when processing the next packet of the message.” Under Implicit’s apparent claim constructions, Decasper98 in view of Decasper97 renders obvious this “state information” element.

Stateful encryption and authentication algorithms are commonplace, and one of ordinary skill in the art would have found it obvious to employ such stateful algorithms in an encryption component and an authentication component. For example, one of ordinary skill in the art would be aware that for both stateful encryption²⁷ and stateful authentication²⁸, a counter is typically

²⁷ See, e.g., Ex. G04 (“Bellare97”) (“A Concrete Security Treatment of Symmetric Encryption” by M. Bellare *et al.*) (August 15, 1997) at 397 (“stateful encryption schemes, in

maintained which influences the encryption or authentication operation, and which is updated each time another operation is performed. Such a counter would comprise “state information” which is **retrieved** each time another packet is to be encrypted or authenticated, used to **perform** the encryption or authentication, and **updated** and stored so it may be used when encrypting or authenticating the next packet.

(b) Claim 15

Because Decasper98 and Decasper97 are similar in approach and detail, Decasper97 confirms the obviousness of claim 15 in a number of ways. One particularly pertinent aspect of Decasper97 is pointed out below.

i. “dynamically identifying a non-predefined sequence”

Claim 15 recites in pertinent part: “dynamically identifying a non-predefined sequence of components for processing each message based on the first packet of the message so that subsequent packets of the message can be processed without re-identifying the components.” Under Implicit’s apparent claim constructions, Decasper98 in view of Decasper97 renders obvious this element.

As explained above, one such obvious sequence would comprise a stateful encryption component and a stateful authentication component. *See* Claim 1 above.

ii. “state information”

which the ciphertext is a function of some information, such as a counter, maintained by the encrypting party and updated with each encryption”). This reference is cited in this context solely to help explain the prior art. *See* MPEP § 2205.

²⁸ *See, e.g.*, Ex. G12 (“Bellare95”) (“XOR MACs: New Methods for Message Authentication Using Finite Pseudorandom Functions” by M. Bellare et al.) (1995) at 16 (“in a stateful [authentication algorithm] the signer maintains information, in our case a counter, which he updates each time a message is signed.”). This reference is cited in this context solely to help explain the prior art. *See* MPEP § 2205.

Claim 15 finally recites in pertinent part: “for each packet of each message, performing the processing of the identified non-predefined sequence of components of the message wherein state information generated by performing the processing of a component for a packet is available to the component when the component processes the next packet of the message.” Under Implicit’s apparent claim constructions, Decasper98 in view of Decasper97 renders obvious this element.

As explained above, it was obvious for the encryption and authentication components to each employ a stateful algorithm wherein a counter would comprise “state information” which is updated during every encryption or authentication operation. *See* Claim 1(ii) above. This “state information” is thus generated by performing the processing of the component, and is available to the component when processing the next packet of the message. *See id.*

(c) Claim 35

Because Decasper98 and Decasper97 are similar in approach and detail, Decasper97 confirms the obviousness of claim 35 in a number of ways. One particularly pertinent aspect of Decasper97 is pointed out below.

i. “dynamically identifying a . . . non-predefined sequence”

Claim 35 recites in pertinent part: “dynamically identifying a message-specific non-predefined sequence of components for processing the packets of each message upon receiving the first packet of the message.” Under Implicit’s apparent claim constructions, Decasper98 in view of Decasper97 renders obvious this element.

As explained above, one such obvious sequence would comprise a stateful encryption component and a stateful authentication component. *See* Claim 1 above.

ii. “state information”

Claim 35 finally recites in pertinent part: “for each packet of the message, invoking the identified non-predefined sequence of components in sequence to perform the processing of each component for the packet wherein each component saves message-specific state information so that that component can use the save message-specific state information when that component performs its processing on the next packet of the message.” Under Implicit’s apparent claim constructions, Decasper98 in view of Decasper97 renders obvious this element.

As explained above, it was obvious for the encryption and authentication components to each employ a stateful algorithm wherein a counter would comprise “state information” which is updated during every encryption or authentication operation. *See* Claim 1(ii) above. This “state information” is thus generated by performing the processing of the component, and is available to the component when processing the next packet of the message. *See id.*

6. Decasper98 in View of Decasper97, Bellare97, and Bellare95 Renders Obvious Claims 1, 15, and 35 Under § 103

The article “A Concrete Security Treatment of Symmetric Encryption” (Ex. 17, “Bellare97”) by M. Bellare *et al.* was published in 1997. The article “XOR MACs: New Methods for Message Authentication Using Finite Pseudorandom Functions” (Ex. 18, “Bellare95”) by M. Bellare et al. was published in 1995. Neither was considered during the prosecution of the ‘163 patent.

If certain aspects recited in claims 1, 15, and 35 of the ‘163 patent are not deemed to be disclosed, inherent, or obvious over Decasper98 in view of Decasper97, then the inclusion of those aspects certainly would be obvious over Decasper98 in view of Decasper97, Bellare97, and Bellare95, under 35 U.S.C. § 103.

It was obvious to supplement the teachings of Decasper98 and Decasper97 with Bellare97 and Bellare95, because Decasper98 and Decasper97 disclose encryption and

authentication operations, and Bellare97 and Bellare95 disclose specific encryption (Bellare97) and authentication (Bellare95) algorithms which might be used.

Decasper98 repeatedly emphasizes the “extensibility” of its platform and expressly declares: “Doubtless, additional plugin types will be introduced by third parties once we have released our code into the public domain.” Ex. 25 at 6, 2, 3, 11. Thus, additional plugins implementing the algorithms of Bellare97 and Bellare95 would be exactly the sort of extensions supported and expected by Decasper98.

(a) Claim 1

i. “a plurality of components”

Claim 1 recites in pertinent part: “providing a plurality of components, each component being a software routine for converting data with an input format into data with an output format.” Under Implicit’s apparent claim constructions, these references render obvious this element.

As explained above, it was obvious that one such sequence would comprise an encryption component and an authentication component. *See* Section V.C.5 (Decasper98+Decasper97) at Claim 1(i) above.

ii. “state information”

Claim 1 finally recites in pertinent part: “for each of a plurality of packets of the message in sequence, for each of a plurality of components in the identified non-predefined sequence, **retrieving** state information relating to performing the processing of the component with the previous packet of the message; **performing** the processing of the identified component with the packet and the retrieved state information; and **storing** state information relating to the processing of the component with the packet for use when processing the next packet of the

message.” Under Implicit’s apparent claim constructions, these references render obvious this “state information” element.

Bellare97 teaches “*stateful* encryption schemes, in which the ciphertext is a function of some information, such as a counter, maintained by the encrypting party and updated with each encryption.” Ex. 17 at 397 (emphasis in original). In its analysis of “some classic symmetric encryption schemes,” Bellare97 concludes that a particular stateful scheme (“stateful XOR, based on a finite PRF”) “has the best security.” *Id.* at 396. “For the stateful XOR scheme we show that . . . this scheme is about as good a scheme as one can possibly hope to get.” *Id.* It was therefore obvious to employ such a stateful algorithm in an encryption component.

Bellare95 teaches “stateful” authentication algorithms in which “the signer maintains information, in our case a counter, which he updates each time a message is signed.” Ex. 18 at 16. In more detail:

In a stateful message authentication scheme, the signer maintains state across consecutive signing requests. (For example, in our counter-based scheme the signer maintains a message counter.) In such a case the signing algorithm can be thought of as taking an additional input—the “current” state *C*, of the signer—and returning an additional output—the signer’s next state.

Id. at 21. Bellare95 analyzes both stateless (“Randomized XOR”) and stateful (“Counter-Based XOR”) authentication algorithms, and observes that “[t]he gain” of the stateful, counter-based algorithm “is greater security.” *Id.* at 22-25 (analysis of stateless), 25-27 (analysis of stateful, counter-based). It was therefore obvious to employ such a stateful algorithm in an authentication component.

The counter used for both stateful encryption and stateful authentication would comprise “state information” which is **retrieved** each time another packet is to be encrypted or

authenticated, used to **perform** the encryption or authentication, and updated and **stored** so it may be used when encrypting or authenticating the next packet.

(b) Claim 15

i. “dynamically identifying a non-predefined sequence”

Claim 15 recites in pertinent part: “dynamically identifying a non-predefined sequence of components for processing each message based on the first packet of the message so that subsequent packets of the message can be processed without re-identifying the components.” Under Implicit’s apparent claim constructions, these references render obvious this element.

As explained above, one such obvious sequence would comprise an encryption component and an authentication component. *See* Section V.C.5 (Decasper98+Decasper97) at Claim 1(i) above.

ii. “state information”

Claim 15 finally recites in pertinent part: “for each packet of each message, performing the processing of the identified non-predefined sequence of components of the message wherein state information generated by performing the processing of a component for a packet is available to the component when the component processes the next packet of the message.” Under Implicit’s apparent claim constructions, these references render obvious this element.

As explained above, it was obvious for the encryption and authentication components to each employ a stateful algorithm wherein a counter would comprise “state information” which is updated during every encryption or authentication operation. *See* Claim 1(ii) above. This “state information” is thus generated by performing the processing of the component, and is available to the component when processing the next packet of the message. *See id.*

(c) Claim 35

i. “dynamically identifying a . . . non-predefined sequence”

Claim 35 recites in pertinent part: “dynamically identifying a message-specific non-predefined sequence of components for processing the packets of each message upon receiving the first packet of the message.” Under Implicit’s apparent claim constructions, these references render obvious this element.

As explained above, it was obvious that one such sequence would comprise an encryption component and an authentication component. *See* Section V.C.5 (Decasper98+Decasper97) at Claim 1(i) above.

ii. “state information”

Claim 35 finally recites in pertinent part: “for each packet of the message, invoking the identified non-predefined sequence of components in sequence to perform the processing of each component for the packet wherein each component saves message-specific state information so that that component can use the save message-specific state information when that component performs its processing on the next packet of the message.” Under Implicit’s apparent claim constructions, these references render obvious this element.

As explained above, it was obvious for the encryption and authentication components to each employ a stateful algorithm wherein a counter would comprise “state information” which is updated during every encryption or authentication operation. *See* Claim 1(ii) above. This “state information” is thus generated by performing the processing of the component, and is available to the component when processing the next packet of the message. *See id.*

7. Decasper98 in View of IBM96 Renders Obvious Claims 1, 15, and 35 Under § 103

The book “Local Area Network Concepts and Products: Routers and Gateways” (Ex. 19, “IBM96”) was published by IBM in May 1996. IBM96 was not considered during prosecution of the ‘163 patent.

If certain aspects recited in claims 1, 15, and 35 of the '163 patent are not deemed to be disclosed, inherent, or obvious over Decasper98 alone, then the inclusion of those aspects certainly would be obvious over Decasper98 in view of IBM96, under 35 U.S.C. § 103.

It was obvious to supplement the teachings of Decasper98 with IBM96 because Decasper98 teaches a general, extensible platform for implementing routers, and IBM96 teaches features which would have been typical of routers of the time period.

(a) Claim 1

i. "a plurality of components"

Claim 1 recites in pertinent part: "providing a plurality of components, each component being a software routine for converting data with an input format into data with an output format." Under Implicit's apparent claim constructions, Decasper98 in view of IBM96 renders obvious this element.

During the pertinent time period, it was commonplace for routers to perform compression on certain traffic being routed through them. This is repeatedly confirmed by IBM96. For example, the "IBM 2210 Nways Multiprotocol Router" could perform "Data Compression over Point-to-Point Protocol" using the "LZ77" compression algorithm. Ex. 19 at 84, 95-96. As another example, IBM96 lists "Data compression" as one of the "Advantages" of its "IBM AnyNet Product Family," explaining that data compression "reduces the amount of data being exchanged between partners, thus improving response time and reducing traffic over the network." *Id.* at 33. Similarly, IBM96 lists "Data compression" one of the "Benefits" of the "2217 Nways Multiprotocol Concentrator" product, explains data compression "[p]rovides higher data rates and improves response times at a lower cost." *Id.* at 200-201.

In view of these various benefits of data compression, it was obvious that in addition to supporting operations such as encryption and authentication, Decasper98 should also support

compression. Decasper98 repeatedly emphasizes the “extensibility” of its platform and expressly declares: “Doubtless, additional plugin types will be introduced by third parties once we have released our code into the public domain.” Ex. 25 at 6, 2, 3, 11. Thus, an additional plugin implementing compression would be exactly the sort of extensions supported and expected by Decasper98.

Since compressed data has a different format from uncompressed data, such a compression component would “convert[] data with an input format into data with an output format.” As combined with other component(s) from Decasper98 (*e.g.*, which perform stateful encryption and/or authentication), such a compression component would comprise “a plurality of components” which read on this claim element.

Since Decasper98 teaches that its plugins components are selected on the basis of separate, independent filter tables, it was obvious that virtually any combination of the components discussed below would be applied to a particular flow, including (as here) the combination of a compression component with other components.

ii. “state information”

Claim 1 finally recites in pertinent part: “for each of a plurality of packets of the message in sequence, for each of a plurality of components in the identified non-predefined sequence, **retrieving** state information relating to performing the processing of the component with the previous packet of the message; **performing** the processing of the identified component with the packet and the retrieved state information; and **storing** state information relating to the processing of the component with the packet for use when processing the next packet of the message.” Under Implicit’s apparent claim constructions, Decasper98 in view of IBM96 renders obvious this “state information” element.

IBM96 discusses and compares the performance of four specific compression algorithms, the top three of which are all “LZ”-based compression algorithms. *See* Ex. 19 at 95-96 (“LZ77” has compression ratio of “2.08:1”; “Stacker-LZS” a ratio of “1.82:1”; “BSD Compress-LZW” a ratio of “2.235:1”; and “Predictor” a ratio of “1.67:1”). Because the top three algorithms discussed by IBM96 are LZ-based and because the “IBM 2210” router specifically uses the “LZ77” algorithm, an LZ-based algorithm such as LZ77 would have been an obvious choice for a compression component to be added to Decasper98. *Id.* at 95-96, 84.

LZ compression algorithms are stateful, and an obvious implementation of them would read on this “state information” claim element.²⁹ Maintaining such state information would entail, for each packet: *e.g.*, **retrieving** the state information, using it to **perform** the compression processing, updating it to reflect the data in the most recent packet, and **storing** it so it can be applied to the next packet.

More generally (and not confined to LZ-based algorithms), stateful (“adaptive”) compression algorithms were commonplace at the time, and obvious implementations of them would likewise read on this “state information” claim element.³⁰

(b) *Claim 15*

i. *“dynamically identifying a non-predefined sequence”*

²⁹ *See, e.g.*, Ex. G09 (“Nelson”) (“The Data Compression Book” by Mark Nelson *et al.*) (1995) at 21 (LZ employs an “adaptive” algorithm which maintains state information in form of, *e.g.*, a sliding “4K-byte window” of the most recent data seen, or an incrementally built dictionary based on *all* of the previously seen data), 18-19. This reference is cited in this context solely to help explain IBM96. *See* MPEP § 2205.

³⁰ *See, e.g.*, Ex. G09 (Nelson) at 18 (“compression research in the last 10 years has concentrated on adaptive models”), 18-19 (including Figures 2.2 and 2.3, showing state information in form of a “Model” which is updated on each new piece of data). This reference is cited in this context solely to help explain IBM96. *See* MPEP § 2205.

Claim 15 recites in pertinent part: “dynamically identifying a non-predefined sequence of components for processing each message based on the first packet of the message so that subsequent packets of the message can be processed without re-identifying the components.” Under Implicit’s apparent claim constructions, Decasper98 in view of IBM96 renders obvious this element.

As explained above, one such obvious sequence would comprise a stateful compression component combined with other component(s) (*e.g.*, a stateful encryption and/or authentication component). *See* Claim 1 above.

ii. “state information”

Claim 15 finally recites in pertinent part: “for each packet of each message, performing the processing of the identified non-predefined sequence of components of the message wherein state information generated by performing the processing of a component for a packet is available to the component when the component processes the next packet of the message.” Under Implicit’s apparent claim constructions, Decasper98 in view of IBM96 renders obvious this element.

As explained above, it was obvious for the compression component to employ a stateful algorithm (such as “LZ77”) whereby state information would be updated during every compression operation. *See* Claim 1(ii) above. This “state information” is thus generated by performing the processing of the component, and is available to the component when processing the next packet of the message. *See id.*

(c) Claim 35

i. “dynamically identifying a . . . non-predefined sequence”

Claim 35 recites in pertinent part: “dynamically identifying a message-specific non-predefined sequence of components for processing the packets of each message upon receiving

the first packet of the message.” Under Implicit’s apparent claim constructions, Decasper98 in view of IBM96 renders obvious this element.

As explained above, one such obvious sequence would comprise a stateful compression component combined with other component(s) (e.g., a stateful encryption and/or authentication component). *See* Claim 1 above.

ii. “state information”

Claim 35 finally recites in pertinent part: “for each packet of the message, invoking the identified non-predefined sequence of components in sequence to perform the processing of each component for the packet wherein each component saves message-specific state information so that that component can use the save message-specific state information when that component performs its processing on the next packet of the message.” Under Implicit’s apparent claim constructions, Decasper98 in view of IBM96 renders obvious this element.

As explained above, it was obvious for the compression component to employ a stateful algorithm (such as “LZ77”) whereby state information would be updated during every compression operation. *See* Claim 1(ii) above. This “state information” is thus generated by performing the processing of the component, and is available to the component when processing the next packet of the message. *See id.*

8. Decasper98 in View of IBM96 and Nelson Renders Obvious Claims 1, 15, and 35 Under § 103

The treatise “The Data Compression Book” (Ex. 5, “Nelson”) by Mark Nelson et al. was published on November 6, 1995. Nelson was not considered during the prosecution of the ‘163 patent.

If certain aspects recited in claims 1, 15, and 35 of the ‘163 patent are not deemed to be disclosed, inherent, or obvious over Decasper98 in view of IBM96, then the inclusion of those

aspects certainly would be obvious over Decasper98 in view of IBM96 and Nelson, under 35 U.S.C. § 103.

It was obvious to supplement the teachings of Decasper98 and IBM96 with Nelson, because IBM96 disclose compression operations performed by routers, and Nelson teaches specific compression algorithms which might be used.

Decasper98 repeatedly emphasizes the “extensibility” of its platform and expressly declares: “Doubtless, additional plugin types will be introduced by third parties once we have released our code into the public domain.” Ex. 25 at 6, 2, 3, 11. Thus, an additional plugin implementing a compression algorithm would be exactly the sort of extension supported and expected by Decasper98.

(a) Claim 1

i. “a plurality of components”

Claim 1 recites in pertinent part: “providing a plurality of components, each component being a software routine for converting data with an input format into data with an output format.” Under Implicit’s apparent claim constructions, these references render obvious this element.

As explained above, one such obvious sequence would comprise a compression component combined with other component(s) (*e.g.*, a stateful encryption and/or authentication component). *See* Section V.C.7 (Decasper98+IBM96) at Claim 1 above.

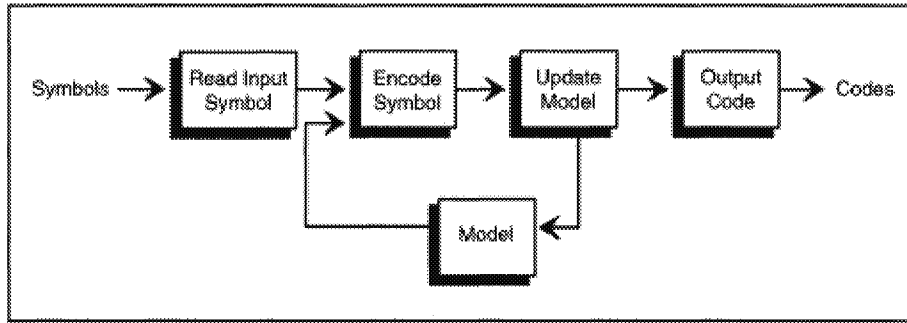
ii. “state information”

Claim 1 finally recites in pertinent part: “for each of a plurality of packets of the message in sequence, for each of a plurality of components in the identified non-predefined sequence, **retrieving** state information relating to performing the processing of the component with the previous packet of the message; **performing** the processing of the identified component with the

packet and the retrieved state information; and **storing** state information relating to the processing of the component with the packet for use when processing the next packet of the message.” Under Implicit’s apparent claim constructions, these references render obvious this “state information” element.

Nelson explains: “Adaptive coding . . . lead[s] to vastly improved compression ratios,” and that “compression research in the last 10 years has concentrated on adaptive models.” Ex. 5 at 8, 18. Adaptive algorithms include such well-known algorithms as “Adaptive Huffman Coding” (chapter 4; *id.* at 75), “Adaptive [Statistical] Modeling” (chapter 6; *id.* at 155), “[Adaptive] Dictionary-Based Compression” (chapter 7: *id.* at 203), and “Sliding Window Compression” (chapter 8; *id.* at 215); and the prominent “LZ” family of compression algorithms (chapter 8 and 9, *id.* at 221, 255). All of these adaptive techniques are lossless, which would be important for accurately transmitting information contained in network packets. *See id.* at 9 (“All of the compression techniques discussed through chapter 9 are ‘lossless’”). In view of the prominence, lossless nature, and improved compression ratios of adaptive algorithms, use of such an algorithm would have been an obvious choice for a compression component.

Nelson further explains the stateful manner in which adaptive coding operates: “When using an adaptive model, data does not have to be scanned once before coding in order to generate statistics [used to perform compression]. Instead, the statistics are **continually modified as new characters are read in and coded**. The general flow of a program using an adaptive model looks something like that shown in Figure[] 2.2” *Id.* at 18 (emphasis added).



Id. at 19 (Figure 2.2: “General Adaptive Compression,” showing “Update Model” (*i.e.*, update state information) after encoding every piece of data). Nelson explains: “adaptive models start knowing essentially nothing about the data” so “when the program first starts it doesn’t do a very good job of compression.” *Id.* at 19. However, “[m]ost adaptive algorithms tend to adjust quickly to the data stream and will begin turning in respectable compression ratios after only a few thousand bytes.” *Id.*

Thus, an obvious implementation of an adaptive algorithm would entail, for each packet, **retrieving** state information, using it to **perform** the compression processing, updating it to reflect the data in the most recent packet, and **storing** it so it can be applied to the next packet.

More narrowly, IBM96 teaches that its “2210” router employs the “LZ77” compression algorithm, so use of that algorithm in particular would have been an obvious design decision over IBM96. *See* Ex. 19 (IBM96) at 95-96, 84. Nelson confirms this algorithm was stateful and “adaptive” in the manner described above. *See, e.g.*, Ex. 5 at 21 (“LZ77” maintains a “dictionary” comprised of, *e.g.*, a sliding “4K-byte window” of the most recently seen data).

(b) *Claim 15*

i. “*dynamically identifying a non-predefined sequence*”

Claim 15 recites in pertinent part: “dynamically identifying a non-predefined sequence of components for processing each message based on the first packet of the message so that

subsequent packets of the message can be processed without re-identifying the components.”

Under Implicit’s apparent claim constructions, these references render obvious this element.

As explained above, one such obvious sequence would comprise a stateful compression component combined with other component(s) (e.g., a stateful encryption and/or authentication component). *See* Claim 1 above.

ii. “state information”

Claim 15 finally recites in pertinent part: “for each packet of each message, performing the processing of the identified non-predefined sequence of components of the message wherein state information generated by performing the processing of a component for a packet is available to the component when the component processes the next packet of the message.”

Under Implicit’s apparent claim constructions, these references render obvious this element.

As explained above, it was obvious for the compression component to employ a stateful, “adaptive” compression algorithm, wherein state information would be updated during every compression operation. *See* Claim 1(ii) above. This “state information” is thus generated by performing the processing of the component, and is available to the component when processing the next packet of the message. *See id.*

(c) Claim 35

i. “dynamically identifying a . . . non-predefined sequence”

Claim 35 recites in pertinent part: “dynamically identifying a message-specific non-predefined sequence of components for processing the packets of each message upon receiving the first packet of the message.” Under Implicit’s apparent claim constructions, these references render obvious this element.

As explained above, one such obvious sequence would comprise a stateful compression component combined with other component(s) (e.g., a stateful encryption and/or authentication component). *See* Claim 1 above.

ii. “state information”

Claim 35 finally recites in pertinent part: “for each packet of the message, invoking the identified non-predefined sequence of components in sequence to perform the processing of each component for the packet wherein each component saves message-specific state information so that that component can use the save message-specific state information when that component performs its processing on the next packet of the message.” Under Implicit’s apparent claim constructions, these references render obvious this element.

As explained above, it was obvious for the compression component to employ a stateful, “adaptive” compression algorithm, wherein state information would be updated during every compression operation. *See* Claim 1(ii) above. This “state information” is thus generated by performing the processing of the component, and is available to the component when processing the next packet of the message. *See id.*

9. Decasper98 in View of RFC 1825, RFC 1829, Decasper97, Bellare97, Bellare95, IBM96, and Nelson Renders Obvious Claims 1, 15, and 35 Under § 103

If certain aspects recited in claims 1, 15, and 35 of the ‘163 patent are not deemed to be disclosed, inherent, suggested, or obvious over Decasper98 alone or in combination with the various grounds of rejection presented above, then the inclusion of those aspects certainly would be obvious over Decasper98 in view of RFC 1825, RFC 1829, Decasper97, Bellare97, Bellare95, IBM96, and Nelson, under 35 U.S.C. § 103, under Implicit’s apparent claim constructions.

All of these references have already been combined with Decasper98 in corresponding sections above, and those sections should be consulted for the detailed manner of applying them

to Decasper98. This section briefly summarizes that material and shows the collective combination of these references would be obvious as well.

Decasper98 teaches a general architecture for router/firewall plugins and repeatedly emphasizes its “extensibility.” Ex. 25 at 1, 2, 3, 11, 6 (“Doubtless, additional plugin types will be introduced by third parties once we have released our code into the public domain.”).

Decasper98 teaches “plugins for IP security,” and **Decasper97** confirms the obviousness of providing separate plugin components for encryption and authentication. **IBM96** confirms the obviousness of an additional plugin component for compression.

Since Decasper98 teaches that its plugin components are selected on the basis of separate, independent filter tables, it was obvious that any two or more of these three types of plugins (encryption, authentication, compression) might be applied to the same flow. This is especially obvious since all three operations would be useful for implementing, *e.g.*, a virtual private network across an expensive link. *See* Ex. 25 (Decasper98) at 5 (“system is configured as entry point into a virtual private network”).

Claims 1, 15, and 35 recite elements regarding “state information.”

RFC 1829 and **Bellare97** confirm the obviousness of employing a stateful encryption algorithm which would read on these elements. **Bellare95** confirms the obviousness of employing a stateful authentication algorithm which would read on these elements. **Nelson** confirms the obviousness of employing a stateful compression algorithm which would read on these elements.

Claim 1 recites each component “being a software routine for converting data with an input format into data with an output format.”

RFC 1825 confirms the obviousness of inserting separate headers into a packet for both encryption and authentication, and this would read on this “converting data” element, under Implicit’s apparent claim constructions. Performing compression on a packet would read on this “converting data” element as well, under Implicit’s apparent claim constructions.

Finally, in addition to the specific plugin components discussed immediately above (encryption, authentication, compression), Decasper98 discloses a number of other plugin components which would read on the “state information” and/or “format” claim elements of claims 1, 15, and 35, including plugin components for IPv6 options, statistics gathering, packet scheduling, and firewall functions. See Sections V.C.1 (Decasper98 102) and V.C.2 (Decasper98 103) above. Since Decasper98 teaches that its plugin components are selected on the basis of separate, independent filter tables, it was obvious for any of these various plugin components to be applied to the same flow as well, in addition to (or instead of) any of the encryption, authentication, or compression components discussed immediately above.

In short, there is no aspect of claims 1, 15, and 35 which was not obvious over the prior art and combinations cited herein.

10. Decasper98 in View of Bellissard Renders Obvious Claims 1, 15, and 35 Under § 103

The article “Dynamic Reconfiguration of Agent-Based Applications” (Ex. 23, “Bellissard”) by Luc Bellissard *et al.* was published by September 10, 1998. Bellissard was not considered during prosecution of the ‘163 patent.

If certain aspects recited in claims 1, 15, and 35 of the ‘163 patent are not deemed to be disclosed, inherent, or obvious over Decasper98 alone, then the inclusion of those aspects certainly would be obvious over Decasper98 in view of Bellissard, under 35 U.S.C. § 103.

It was obvious to supplement the teachings of Decasper98 with Bellissard because Decasper98 teaches an extensible architecture for implementing firewalls and routers, and Bellissard teaches a technique for enhancing the dynamic extensibility of such an architecture.

While Decasper98 already teaches a platform wherein an administrator can dynamically add and configure components “even when network traffic is transiting through the system” (Ex. 25 at 9), Bellissard provides additional detail on how such a system could operate and on another way in which it could be implemented.

(a) *Claim 1*

i. “*dynamically identifying a non-predefined sequence*”

Claim 1 recites in pertinent part: “for the first packet of the message, dynamically identifying a non-predefined sequence of components for processing the packets of the message such that the output format of the components of the non-predefined sequence match the input format of the next component in the non-predefined sequence.” Under Implicit’s apparent claim constructions, Decasper98 in view Bellissard renders obvious this element.

Decasper98 alone renders obvious these elements. *See* Section V.C.2 (Decasper98 103) at Claim 1. As applied to Decasper98, Bellissard further underscores the “dynamic[.]” nature of the identification, under Implicit’s apparent claim constructions, as explained below.

Bellissard teaches a technique for “dynamically modifying” and “[d]ynamically reconfiguring” an application while the application is *still operating*, without halting the application in order to reconfigure it. Ex. 23 at 1-3. Bellissard explains the motivation for this technique is that “new functionalities” may be “required by the users” at any time:

Reconfiguration is thus an answer to the problems of dynamically modifying the application architecture (both in terms of agent functions and of the sequence of actions to be performed), while the application is operating. This cannot be achieved with current techniques such as configuration of predefined parameters, because

it is impossible to predict all the new functionalities that can be required by the users.

Id. at 2.

It was particularly obvious to apply the technique of Bellissard to the extensible router/firewall architecture of Decasper98, because a “firewall” is precisely the example chosen by Bellissard of “a typical full-size application” which would “emphasize the benefits of” the Bellissard technique. *Id.* at 1; Ex. 25 (Decasper98) at 2 (“Our framework is also very well suited to . . . security devices like Firewalls”). It was further obvious to apply the Bellissard technique of “dynamic reconfiguration” to Decasper98, because Decasper98 repeatedly emphasizes that the “extensibility” of its architecture which permits new components to be “dynamically loaded at run time.” *E.g.*, Ex. 25 at 2 (“Extensibility: New plugins can be dynamically loaded at run time”), 3 (“The primary goal of our proposed architecture was to build a modular and extensible networking subsystem that supported the concept of flows,” including “Dynamic loading and unloading of plugins at run time into the networking subsystem.”).

The “dynamic reconfiguration” of technique Bellissard includes performing the following two operations “while the application is operating”: (1) “Modifying the architecture of an application (adding/removing modules, and modifying the interconnection pattern)”; and (2) “Modifying the implementation of a component.” Ex. 23 at 2.

As applied to Decasper98, the first operation (“Modifying the architecture of an application” including “adding/removing modules”) would clearly encompass adding or removing certain “plugin” modules of Decasper98 while the system of Decasper98 was still operating. *See* Ex. 23 at 2; Ex. 25 (Decasper98) at 2 (“New plugins can be dynamically loaded at run time.”), 6 (“Doubtless, additional plugin types will be introduced by third parties once we have released our code into the public domain.”). Bellissard explains “it is impossible to predict

all the new functionalities that can be required by users.” Ex. 23 at 2. In the context of the extensible router/firewall architecture of Decasper98, providing the required “new functionalities” would typically entail the provision of new Decasper98 plugins: *e.g.*, to support a new IPv6 option functionality, a new authentication functionality, a new compression functionality, and so on. Indeed, Bellissard specifically teaches the insertion of a new “compression” component into a firewall system while it is still operating. Ex. 23 at 2 (“insertion of a compression agent”). Using the Bellissard technique, such new plugins could be “dynamically” added to Decasper98 while Decasper98 was still operating—with the advantage that flows could begin to take advantage of the new functionalities immediately, and without disrupting existing flows through the system. *See* Ex. 23 at 1-2.³¹

As applied to Decasper98, the second operation (“Modifying the implementation of a component”) would clearly encompass modifying the implementation of a “plugin” module of Decasper98 while the system of Decasper98 was still operating. *See* Ex. 23 at 2, Ex. 25 at 2. For example, a more efficient, higher-performance implementation might become available for an authentication plugin, or an encryption plugin, or a compression plugin, and so on. Using the Bellissard technique, such a plugin could be “dynamically modified” to employ the new, more efficient implementation while Decasper98 was still operating—with the advantage that the plugin could begin to take advantage of the improved implementation immediately, and without disrupting existing flows.

Application of the above techniques to Decasper98 would be an especially straightforward task, because unlike more “monolithic” prior art routers and firewalls,

³¹ Again, Decasper98 already teaches substantially this same technique, but Bellissard provides additional detail on how such a system could operate and on another way in which it could be implemented. *See* Ex. 25 (Decasper98) at 9 (adding and configuring a plugin “even when network traffic is transiting through the systems”).

Decasper98 had been specifically architected to divide its various functions into discrete “plugins” which were “modular, “extensible,” and could be “dynamically loaded at runtime.” See e.g., Ex. 25 at 1-2.

To summarize, the combination of Decasper98 and Bellissard renders obvious a system in which the plugin components of Decasper98 could be dynamically modified or dynamically added at any moment during runtime--while the system was still operating—and could thereby take advantage of the newly added or modified components. Under Implicit’s apparent claim constructions, such a system would clearly read on “dynamically identifying a non-predefined sequence of components for processing the packets of the message.”

(b) Claim 15

i. “dynamically identifying a non-predefined sequence”

Claim 15 recites in pertinent part: “dynamically identifying a non-predefined sequence of components for processing each message based on the first packet of the message so that subsequent packets of the message can be processed without re-identifying the components.” Under Implicit’s apparent claim constructions, Decasper98 in view of Bellissard renders obvious this element. See Claim 1 above.

(c) Claim 35

i. “dynamically identifying a . . . non-predefined sequence”

Claim 35 recites in pertinent part: “dynamically identifying a message-specific non-predefined sequence of components for processing the packets of each message upon receiving the first packet of the message.” Under Implicit’s apparent claim constructions, Decasper98 in view of Bellissard renders obvious this element. See Claim 1 above.

11. Decasper98 in View of Fraser Renders Obvious Claims 1, 15, and 35 Under § 103

The publication “DTE Firewalls: Phase Two Measurement and Evaluation Report” (Ex. 24, “Fraser”) by Timothy J. Fraser *et al.* was published by Trusted Information Systems on July 22, 1997. Fraser was not considered during prosecution of the ‘163 patent.

If certain aspects recited in claims 1, 15, and 35 of the ‘163 patent are not deemed to be disclosed, inherent, or obvious over Decasper98 alone, then the inclusion of those aspects certainly would be obvious over Decasper98 in view of Fraser, under 35 U.S.C. § 103.

It was obvious to supplement the teachings of Decasper98 with Fraser because Decasper98 teaches an extensible architecture for implementing firewalls and routers, and Fraser teaches a technique for enhancing the dynamic configurability of such an architecture.

While Decasper98 already teaches a platform wherein an administrator can dynamically configure policies (expressed in filters) “even when network traffic is transiting through the system” (Ex. 25 at 9), Fraser teaches a more comprehensive framework for such a capability, and provides additional detail on how such a framework would be implemented.

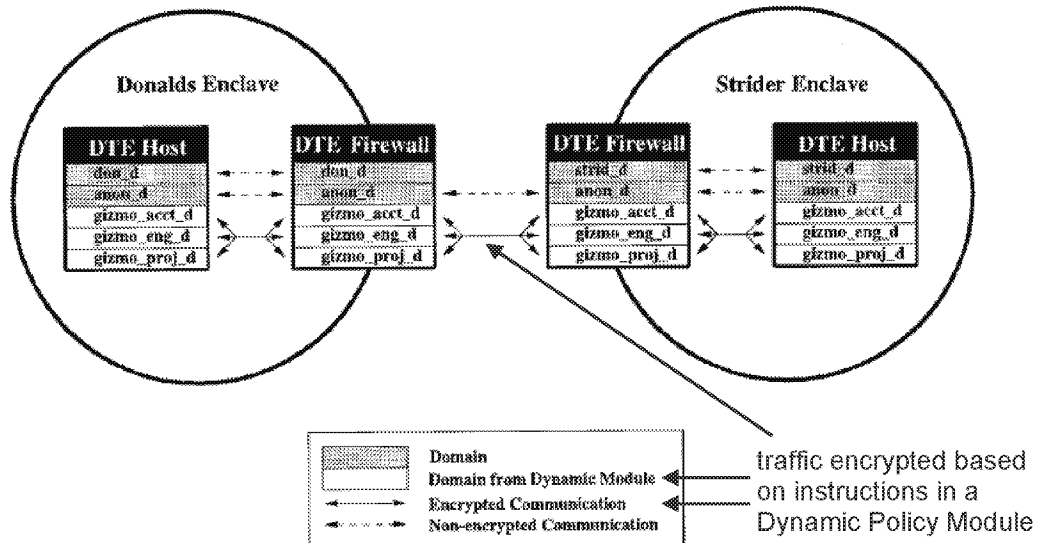
(a) Claim 1

i. “dynamically identifying a non-predefined sequence”

Claim 1 recites in pertinent part: “for the first packet of the message, dynamically identifying a non-predefined sequence of components for processing the packets of the message such that the output format of the components of the non-predefined sequence match the input format of the next component in the non-predefined sequence.” Under Implicit’s apparent claim constructions, Decasper98 in view Fraser renders obvious this element.

Decasper98 alone renders obvious these elements. *See* Section V.C.2 (Decasper98 103) at Claim 1. As applied to Decasper98, Fraser further underscores the “dynamic[]” nature of the identification, under Implicit’s apparent claim constructions, as explained below.

Fraser teaches “Dynamic Policy Modules” which an administrator uses to control the behavior of a firewall: *e.g.*, these modules define which traffic flowing through the firewall should be encrypted, and which network destinations should be accessible to which users. Ex. 24 at 10, 6-7.



Id. at 7 (Figure 3, showing encryption performed according to instructions in “Dynamic Module[s]”; “The transient domains originating in dynamic modules are not shaded.”).

Fraser explains that before Dynamic Policy Modules were introduced, “the primary method” for an administrator to alter a firewall’s “security policy” was “to edit the policy specification and reboot the kernel for the updated policy to take effect.” *Id.* at 8. This approach was “impractical for operational systems,” because “[r]estructuring the policy and rebooting kernels for each change would result in an undesirable and impractical loss of service.” *Id.* at 9.

Dynamic Policy Modules address this “undesirable and impractical” situation by allowing administrators to make minor or major alterations to a firewall’s policies *without* rebooting the device:

The main contribution of dynamic policy module support . . . is increased functionality. As described in section 2.1.2, dynamic

policy modules provide administrators with an organized framework for managing policy change. Administrators can use dynamic policy modules to specify the policy governing new activities and trust relationships. They may add policy support for a new activity or trust relationship to a [firewall] kernel by loading the appropriate module. Similarly, they can remove the support by unloading the module. Administrators may load and unload modules as the kernel runs. The ability to dynamically reconfigure a kernel's policy as it runs allows administrators to add and remove policy support for trust relationships without requiring system down-time and the resulting disruption of service availability. This method of policy configuration is superior to the [previous] method, which involved modifying a kernel's base policy description and then rebooting the kernel.

Id. at 37.

Rather than being narrowly confined to controlling one or two policy options, Dynamic Policy Modules provide a “wide-ranging ability” to change many aspects of a firewall’s policies.

See id. at 19.

Once made available, Dynamic Policy Modules become the primary means for administrators to modify a firewall’s policies: “Dynamic policy modules are the atomic unit of policy change. Typically, when administrators need to extend a policy to govern a new activity, they will encapsulate the extension in a dynamic policy module.” *Id.* at 12.

It was obvious to apply the Dynamic Policy Modules framework of Fraser to Decasper98, in order to provide a more comprehensive framework³² for avoiding any “undesirable and impractical” need to reboot the Decasper98 device under any circumstances. *See id.* at 9.

Decasper98 was an especially obvious candidate for this technique, because Fraser uses the technique to control the policies of “application gateway firewall[s],” and Decasper98 teaches an

³² Again, Decasper98 already teaches substantially this same technique of modifying the system’s configured policies while the system is operating, but Fraser teaches a more comprehensive framework for such a capability, and provides additional detail on how such a framework would be implemented. *See Ex. 25 (Decasper98)* at 9 (adding and configuring a new plugin “even when network traffic is transiting through the system”).

architecture that is “very well suited to Application Layer Gateways . . . and to security devices like Firewalls.” *Id.* at 6; Ex. 25 at 2.

As applied to Decasper98, Dynamic Policy Modules would allow an administrator to modify the policies which determine which plugins are assigned to which flows. *See* Ex. 25 (Decasper98) at 7. The parallels between the two systems are particularly clear on this point. For example, Fraser’s Dynamic Policy Modules control, *e.g.*, which traffic is encrypted, and Decasper98’s policies (expressed in filters) control, *e.g.*, which flows are encrypted by an encryption plugin. Ex. 24 at 7, Ex. 25 at 5-7.

Application of the above techniques to Decasper98 would be a straightforward task, because unlike more “monolithic” prior art routers and firewalls, Decasper98 had been specifically architected to divide its various functions into discrete “plugins” which were “modular, “extensible,” and could be “dynamically loaded at runtime.” *See e.g.*, Ex. 25 at 1-2. Moreover, Decasper98 is already architected to permit changes to its configured policies (filters) “even when network traffic is transiting through the system.” *Id.* at 9.

To summarize, the combination of Decasper98 and Fraser renders further obvious a system in which the policies determining the identified sequence of plugin components could be dynamically modified or dynamically added at any moment during runtime—while the system was still operating. Under Implicit’s apparent claim constructions, such a system would clearly read on “dynamically identifying a non-predefined sequence of components for processing the packets of the message.”

(b) Claim 15

i. “dynamically identifying a non-predefined sequence”

Claim 15 recites in pertinent part: “dynamically identifying a non-predefined sequence of components for processing each message based on the first packet of the message so that

subsequent packets of the message can be processed without re-identifying the components.”

Under Implicit’s apparent claim constructions, Decasper98 in view of Fraser renders obvious this element. *See* Claim 1 above.

(c) *Claim 35*

i. “*dynamically identifying a . . . non-predefined sequence*”

Claim 35 recites in pertinent part: “dynamically identifying a message-specific non-predefined sequence of components for processing the packets of each message upon receiving the first packet of the message.” Under Implicit’s apparent claim constructions, Decasper98 in view of Fraser renders obvious this element. *See* Claim 1 above.

12. Decasper98 in View of RFC 1825, RFC 1829, RFC 1883, Huitema, Decasper97, Bellare97, Bellare95, IBM96, Nelson, Bellissard, and Fraser Renders Obvious Claims 1, 15, and 35 Under § 103

If certain aspects recited in claims 1, 15, and 35 of the ‘163 patent are not deemed to be disclosed, inherent, suggested, or obvious over Decasper98 alone or in combination with the various grounds of rejection presented above, then the inclusion of those aspects certainly would be obvious over Decasper98 in view of RFC 1825, RFC 1829, RFC 1883, Huitema, Decasper97, Bellare97, Bellare95, IBM96, Nelson, Bellissard, and Fraser under 35 U.S.C. § 103, under Implicit’s apparent claim constructions.

All of these references have already been combined with Decasper98 in corresponding sections above, and those sections should be consulted for the detailed manner of applying them to Decasper98. This section briefly summarizes that material and shows the collective combination of these references would be obvious as well.

Decasper98 teaches a general architecture for router/firewall plugins and repeatedly emphasizes its “extensibility.” Ex. 25 at 1, 2, 3, 11, 6 (“Doubtless, additional plugin types will be introduced by third parties once we have released our code into the public domain.”).

Decasper98 teaches “plugins for IP security,” and **Decasper97** confirms the obviousness of providing separate plugin components for encryption and authentication. **IBM96** confirms the obviousness of an additional plugin component for compression. Decasper98 also teaches “plugins implementing IPv6 options.” *Id.* at 4.

Since Decasper98 teaches that its plugin components are selected on the basis of separate, independent filter tables, it was obvious that any two or more of these four types of plugins (encryption, authentication, compression, IPv6 options) might be applied to the same flow. This is especially obvious since the first three operations would be useful for implementing, *e.g.*, a virtual private network across an expensive link, and IPv6 options are of general usefulness. *See* Ex. 25 (Decasper98) at 5 (“system is configured as entry point into a virtual private network”).

Claims 1, 15, and 35 recite elements regarding “state information.”

RFC 1829 and **Bellare97** confirm the obviousness of employing a stateful encryption algorithm which would read on these elements. **Bellare95** confirms the obviousness of employing a stateful authentication algorithm which would read on these elements. **Nelson** confirms the obviousness of employing a stateful compression algorithm which would read on these elements. **RFC 1883** confirms the obviousness of employing a stateful algorithm for implementing IPv6 options which would read on these elements.

Claim 1 recites each component “being a software routine for converting data with an input format into data with an output format.”

RFC 1825 confirms the obviousness of inserting separate headers into a packet for both encryption and authentication, and this would read on this “converting data” element, under Implicit’s apparent claim constructions. Performing compression on a packet would read on this “converting data” element as well, under Implicit’s apparent claim constructions. **Huitema**

confirms the obviousness of adding or removing headers while processing IPv6 options, which would read on the “converting data” element as well, under Implicit’s apparent claim constructions.

In addition to the specific plugin components discussed immediately above (encryption, authentication, compression, IPv6 options), Decasper98 discloses a number of other plugin components which would read on the “state information” and/or “format” claim elements of claims 1, 15, and 35, including plugin components for statistics gathering, packet scheduling, and firewall functions. *See* Sections V.C.1 (Decasper98 102) and V.C.2 (Decasper98 103) above. Since Decasper98 teaches that its plugin components are selected on the basis of separate, independent filter tables, it was obvious for any of these various plugin components to be applied to the same flow as well, in addition to (or instead of) any of the encryption, authentication, compression, or IPv6 options components discussed immediately above.

Claims 1, 15, and 35 recite “dynamically identifying a . . . non-predefined sequence of components.”

Decasper98 selects the sequence of plugin components for a flow on the basis of *multiple independent filters*, which “even with very few installed filters” leads to “exponentially” many valid component sequences—so many, in fact, that it is “infeasible” to even list them in memory ahead of time. Ex. 25 at 7. Decasper98 therefore adopts an algorithmic approach, of dynamically generating the sequence when the first packet of a flow arrives, by applying its multiple independent filters to the packet data which did not exist in the system until the packet arrived. Under Implicit’s apparent claim constructions, this technique alone reads on these “dynamic[]” claim elements.

Moreover, **Decasper98** also teaches that new plugin components may be added and configured by an administrator at runtime, “even when network traffic is transiting through the system”—including at least up to the very moment before a new flow would begin. *Id.* at 9. This also reads on these “dynamic[]” claim elements, under Implicit’s apparent claim constructions.

Like Decasper98, **Bellissard** teaches dynamically adding new components while the system is operating. It provides additional detail on how such a system could operate, and on another way in which it could be implemented. Bellissard further teaches the dynamic modification of existing components—again, while the system is operating. Under Implicit’s apparent claim constructions, both of these techniques would read on these “dynamic[]” claim elements.

Like Decasper98, **Fraser** teaches dynamically configuring firewall policies while the system is operating. It teaches a more comprehensive framework for this capability, and details another manner in which it could be implemented. Under Implicit’s apparent claim constructions, such dynamic configuration of policies would read on these “dynamic[]” claim elements.

In short, there is no aspect of claims 1, 15, and 35 which was not doubly or triply obvious over the prior art and combinations cited herein.

D. Mosberger (Exhibit 31)

“Scout: A Path-Based Operating System,” is a dissertation submitted by David Mosberger to the faculty of the Department of Computer Science at The University of Arizona (Exhibit 31, “Mosberger”). It was published in 1997.

Mosberger was previously considered by the PTO during *ex parte* reexamination proceedings, during which time the PTO initially sustained multiple anticipation rejections over

Mosberger, concluding that each and every element of claims 1, 15, and 35 was disclosed in the reference. However, after a number of exchanges with the patentee, the PTO ultimately accepted the patentee's arguments that one of the claim limitations was missing—"dynamically identifying"—and issued a notice of intent to issue a reexamination certificate based on the patentee's assertion that "Mosberger does not dynamically identify sequences of components."

Notwithstanding this prior consideration, Mosberger is presented here in a new light in at least two ways. First, a more detailed explanation is provided regarding the "dynamic" nature of Mosberger, contrary to the representations of the patentee during *ex parte* reexamination. Second, to the extent Mosberger is still deemed as lacking adequate disclosure on the "dynamic" limitation, a number of obviousness combinations of Mosberger and other references are provided below, which clearly satisfy any purported deficiency in Mosberger. The PTO has not previously considered any of these combinations.

In any event, the finding of a "reasonable likelihood" under Section 312 is "not precluded by the fact that a patent or printed publication was previously cited by or to the Office or considered by the Office." 35 U.S.C. § 312(a).

1. Mosberger Anticipates Claims 1, 15, and 35 Under § 102(a) and (b)

(a) Claim 1

i. "A method . . . for processing a message"

Claim 1 recites: "A method in a computer system for processing a message having a sequence of packets, the method comprising" Under Implicit's apparent claim constructions, Mosberger discloses this element.

Mosberger discloses that Scout is an "operating system architecture that is designed specifically to accommodate the needs of communication-centric systems." Ex. 31 at 13. Scout

uses paths for processing a “sequence of network packets” and uses demultiplexing to “find the appropriate path for a given message.” *Id.* at 36, 85.

ii. “a plurality of components . . .”

Claim 1 further recites: “providing a plurality of components, each component being a software routine for converting data with an input format into data with an output format.”

Under Implicit’s apparent claim constructions, Mosberger discloses this element.

Mosberger discloses a “plurality of components” for processing messages. For example, Mosberger describes the use of paths to process a dataflow through a series of software modules. *See Ex. 31 at 36; see also id.* Figure 2.4. Modules are “entities that route data through the system.” *Id.* at 61. Some “typical examples are modules that implement networking protocols, such as IP, UDP, or TCP.” *Id.* at 62. Mosberger describes series of modules as a “linear sequence of components.” *Id.* at 57.

Mosberger’s paths are capable of “converting data with an input format into data with an output format.” Generally speaking, a path is “a dataflow that starts at a source device and ends at a destination device.” *Id.* at 36. A series of “modules” makes up a path. In a path, “each module processes data in a well-defined manner.” *Ex. 31 at 37.* As a result, paths can, for example, “transform a series of network packets into a sequence of SCSI disk blocks.” *Id.* at 36. Figure 2.4 of Mosberger “shows a sample path as a bold line starting at module FDDI, passing through IP, UDP, MFLOW, MPEG, WiMP, and finally arriving at DISPLAY.” *Id.* at 37; *see also id.* Figure 2.4.

iii. “dynamically identifying a non-predefined sequence”

Claim 1 further recites: “for the first packet of the message, dynamically identifying a non-predefined sequence of components for processing the packets of the message such that the output format of the components of the non-predefined sequence match the input format of the

next component in the non-predefined sequence, wherein dynamically identifying includes selecting individual components to create the non-predefined sequence of components after the first packet is received.” Under Implicit’s apparent claim constructions, Mosberger discloses this element.

Mosberger discloses “dynamically identifying” under Implicit’s apparent claim construction. Mosberger identifies two possible approaches to path creation: (a) “paths are pre-specified” or (b) “paths are created (discovered) incrementally.” Ex. 31 at 39. Mosberger rejects the “pre-specifying paths” approach – *i.e.*, a system that “provide[s] a table that translates the properties of the desired path into a sequence of modules that the path needs to traverse to satisfy these properties” – in favor of a system that “create[s] paths incrementally.” *Id.* at 40. This is because “[i]n many cases it is beneficial to exploit information that is available at runtime only. For this reason, paths need to be created and destroyed dynamically at runtime.” *Id.* at 39. As Mosberger explains, “*runtime* covers all the steps that occur after the system has been booted on the target machine. During that time, paths *may be created*, used, and destroyed.” *Id.* at 61; *see also id.* Figure 3.1.

Mosberger also makes clear that path changes can happen during runtime. For example, Mosberger explains that “a command-line interpreter is likely to create a path to the input device (e.g., the keyboard) during initialization.” Ex. 31 at 47. “New paths” can then be created through the “handling of key-strokes.” *Id.*

Moreover, given Implicit’s apparent claim constructions, Mosberger expressly illustrates that the Scout system can make “dynamic routing decision[s]”:

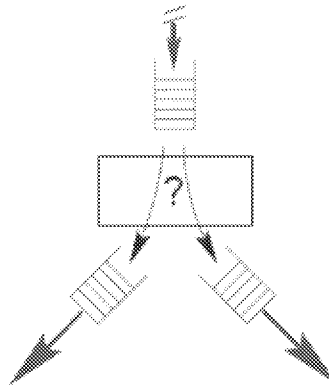


Figure 2.5: Dynamic Routing Decision

Id. at 42 (emphasis in original).

The components disclosed in Mosberger are also used in a manner “such that the output format of the components . . . match the input format of the next component,” under Implicit’s apparent claim construction. *See* Section IV(C). In Mosberger’s system, “a data-item arrives at the input queue, the path is scheduled for execution, and the *transformed data* is deposited in an output queue.” Ex. 31 at 48. As explained previously, data may be processed by multiple components (or modules) in the course of moving through a path. *See id.* at 36; *see also* Figure 2.4. Because packets compatibly move from component to component, this element is satisfied under Implicit’s apparent claim construction.

The “dynamically identifying” as disclosed in Mosberger (under Implicit’s apparent claim construction) also “includes selecting individual components to create the non-predefined sequence of components after the first packet is received.” In Mosberger’s system, individual modules can “make a *dynamic* routing decision” to ensure that data is “processed appropriately.” Ex. 31 at 41-42; *see also* Figure 2.5. This “dynamic routing decision” is “based on the contents of the data being communicated.” *Id.* at 88. It is designed to be able “to exploit information that is available at runtime only.” *Id.* at 36. As Mosberger explains:

path creation is initiated at the module that is to form one end of the path. This module uses the invariants to make a routing decision, that is, a decision as to which module a path with the specified invariants must traverse next. Path creation is then forwarded to that next module. This process repeats itself until either there is no next module (i.e., the edge of the module graph has been reached) or until a module is reached that, based on the specified invariants, cannot make a definite routing decision. As part of making a routing decision, **a module is free to update the invariants since new invariants may become available in that module or old invariants may be invalid beyond that module.**

Id. at 40.

iv. “storing an indication of . . . the identified components”

Claim 1 further recites “storing an indication of each of the identified components so that the non-predefined sequence does not need to be re-identified for subsequent packets of the message.” Under Implicit’s apparent claim constructions, Mosberger discloses this element.

As explained above, paths form when modules make “dynamic routing decision[s]” that are “based on the contents of the data being communicated.” Ex. 31 at 41-41, 88. “Stages” along the path “provide a place to store information that is path-specific, but private to the modules.” *Id.* at 73; *see also id.* Figure 3.5. Once a path is formed, the “sequence of modules being traversed is known and fixed for the lifetime of a path.” *Id.* at 54. To be known and fixed, the sequence of modules for any given path must be stored as claimed in the patent.

v. “state information”

Claim 1 further recites “for each of a plurality of packets of the message in sequence, for each of a plurality of components in the identified non-predefined sequence, **retrieving** state information relating to performing the processing of the component with the previous packet of the message; **performing** the processing of the identified component with the packet and the retrieved state information; and **storing** state information relating to the processing of the

component with the packet for use when processing the next packet of the message.” Under Implicit’s apparent claim constructions, Mosberger discloses these “state information” elements.

Implicit has taken a broad view of the “state information” limitations, arguing that they cover the retrieval, use, and storage of the identified sequence of components (*e.g.*, a flow record) after the first packet is received. *See* Section IV(C). Mosberger’s Scout retrieves, uses, and stores flow records in this manner to facilitate processing of packets in the same message after the first packet is received and a flow entry built. For example, Scout uses “threads” to move data in paths. *See* Ex. 31 at 100-104. “[T]hreads inherit the scheduling parameters (policy and priority) from the path they are executing in.” *Id.* 102. These parameters inform the thread what priority it has in the path vis-à-vis other threads and must therefore be stored to make inheritance possible or to preserve the thread when it does not have priority. *See id.* at 100-104. For packets traversing the same path, threads retrieve and apply the same scheduling parameters to ensure that packets of the same flow receive similar treatment.

(b) Claim 15

i. “demultiplexing packets of messages”

Claim 15 recites: “A method in a computer system demultiplexing packets of messages.” Under Implicit’s apparent claim constructions, Mosberger discloses this element. *See* Claim 1(i) (showing “A method in a computer system for processing a message having a sequence of packets”) above. Under Implicit’s apparent claim constructions, “demultiplexing” a packet is satisfied by routing a packet to the correct sequence of components for processing it—and Mosberger performs this function. *See* Section IV(C) and, *e.g.*, Claim 1(iii) (showing “dynamically identifying a non-predefined sequence of components for processing the packets of the message”) above. In any event, Mosberger contains an elaborate discussion of

demultiplexing, including an entire section of a chapter on Scout architecture devoted to the subject (Section 3.4, entitled “Demultiplexing”). Ex. 31 at 85-99.

ii. “dynamically identifying a non-predefined sequence”

Claim 15 further recites “dynamically identifying a non-predefined sequence of components for processing each message based on the first packet of the message so that subsequent packets of the message can be processed without re-identifying the components.” Under Implicit’s apparent claim constructions, Mosberger discloses this element. *See* Claim 1(iii) (showing “for the first packet of the message, dynamically identifying a non-predefined sequence of components for processing the packets of the message”) and Claim 1(iv) (showing “storing an indication of each of the identified components so that the non-predefined sequence does not need to be re-identified for subsequent packets of the message”) above.

iii. “different . . . sequences of components can be identified”

Claim 15 further recites “different non-predefined sequences of components can be identified for different messages.” Under Implicit’s apparent claim constructions, Mosberger discloses this element. Mosberger expressly illustrates how different messages can be handled with different sequences of components:

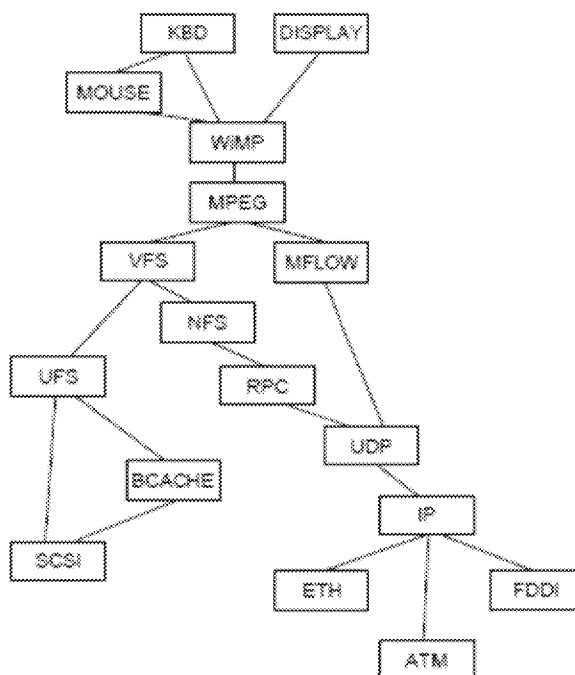


Figure 2.2: Example Modular System

Id. at 35; *see also* Claim 1(ii) and (iii) (showing “plurality of components” and “dynamically identifying includes selecting individual components to create the non-predefined sequence of components”) above.

iv. “each component being a software routine”

Claim 15 further recites “each component being a software routine.” Under Implicit’s apparent claim constructions, Mosberger discloses this element. *See* Claim 1(ii) (showing “each component being a software routine”) above.

v. “selecting individual components”

Claim 15 further recites “dynamically identifying includes selecting individual components to create the non-predefined sequence of components.” Under Implicit’s apparent claim constructions, Mosberger discloses this element. *See* Claim 1(iii) (showing “dynamically identifying includes selecting individual components to create the non-predefined sequence of components”) above.

vi. *“state information”*

Claim 15 finally recites “for each packet of each message, performing the processing of the identified non-predefined sequence of components of the message wherein state information generated by performing the processing of a component for a packet is available to the component when the component processes the next packet of the message.” Under Implicit’s apparent claim constructions, Mosberger discloses this element. *See* Claim 1(v) (showing similar “state information” element) above.

(c) *Claim 35*

i. *“instructions for demultiplexing packets of messages”*

Claim 35 recites: “A computer-readable medium containing instructions for demultiplexing packets of messages.” Under Implicit’s apparent claim constructions, Mosberger discloses this element. Mosberger makes the design choice to implement its system “using a relatively low-level language such as C” which is then translated into “machine code.” *Id.* at 71. One of ordinary skill would have understood that the machine code would be stored and executed on a physical computer-readable medium. *See also* Claim 15(i) showing (“demultiplexing packets of messages”) above.

ii. *“dynamically identifying a . . . non-predefined sequence”*

Claim 35 recites: “dynamically identifying a message-specific non-predefined sequence of components for processing the packets of each message upon receiving the first packet of the message.” Under Implicit’s apparent claim constructions, Mosberger discloses this element. *See* Claim 1(iii) (showing “for the first packet of the message, dynamically identifying a non-predefined sequence of components for processing the packets of the message”) above.

iii. *“subsequent packets . . . can use the . . . sequence”*

Claim 35 further recites that “subsequent packets of the message can use the message-specific non-predefined sequence identified when the first packet was received.” Under Implicit’s apparent claim constructions, Mosberger discloses this element. *See* Claim 1(iv) (showing “storing an indication of each of the identified components so that the non-predefined sequence does not need to be re-identified for subsequent packets of the message”) above.

iv. “selecting individual components”

Claim 35 further recites that “dynamically identifying includes selecting individual components to create the message-specific non-predefined sequence of components.” Under Implicit’s apparent claim constructions, Mosberger discloses this element. *See* Claim 1(iii) (showing “dynamically identifying includes selecting individual components to create the non-predefined sequence of components after the first packet is received”) above.

v. “state information”

Claim 35 further recites “for each packet of the message, invoking the identified non-predefined sequence of components in sequence to perform the processing of each component for the packet wherein each component saves message-specific state information so that that component can use the save message-specific state information when that component performs its processing on the next packet of the message.” Under Implicit’s apparent claim constructions, Mosberger discloses this element. *See* Claim 1(v) (showing similar “state information” element) above.

2. Mosberger Renders Obvious Claims 1, 15, and 35 Under § 102(a), (b)

If certain aspects recited in claims 1, 15, and 35 of the ‘163 patent are not deemed to be disclosed or inherent over Mosberger, then the inclusion of those aspects certainly would be obvious over Mosberger in light of the background knowledge of one of ordinary skill in the art, under 35 U.S.C. § 103.

Mosberger discloses all of the limitations of Claims 1, 15, and 35— including the “dynamically identifying” limitation—for the reasons set forth above. However, even if Mosberger is deemed not to have an express disclosure of the “dynamically identifying” limitation, one of ordinary skill in the art would have immediately appreciated that the system disclosed in Mosberger could have been modified without difficulty to include such functionality.

Specifically, during prior *ex parte* reexamination of the ‘163 patent, focus was placed on the passage of Mosberger at page 71 to the effect that “the Scout module graph is presently configured at build time and, hence, it is not possible to extend the graph at runtime.” Ex. 31 at 71. However, Mosberger goes on to expressly state “However, it is *straight-forward* to *add a dynamic module-loading facility* to Scout.” *Id.* Mosberger expresses further confidence in the ease with which such a “dynamic loading” functionality could be added to the disclosed Scout system, stating that the “actual dynamic loading” is *not* the “biggest issue” in modifying Scout, but rather “the security issue.” *Id.* And, of course, the claims of the ‘163 patent contain no limitation directed to any such “security issue”; in other words, even an insecure implementation of “dynamic loading” would satisfy the “dynamically identifying” limitation of the ‘163 patent.

Furthermore, Mosberger proposes yet another modification of Scout that would permit “dynamically identifying,” which is “to configure a virtual machine module into the graph that would allow interpreted code to be downloaded and executed inside Scout.” *Id.* For example, Mosberger here drops a reference to footnote 39, which directs the reader to a reference entitled “The Java Application Programming Interface.” Ex. 31 at 71, 167. One of ordinary skill in the art would have appreciated that the Java programming environment can readily provide a

“virtual machine” to be used to permit code to be dynamically “downloaded and executed inside Scout.” *Id.*

3. Mosberger in View of HotLava Renders Obvious Claims 1, 15, and 35 Under § 103

The article “Implementing Communication Protocols in Java” (Ex. 32, “HotLava”) by Bobby Krupczak *et. al* was published in October 1998. It describes a Java-based protocol subsystem called “HotLava.” HotLava was not considered during the prosecution of the ‘163 patent.

If certain aspects recited in claims 1, 15, and 35 of the ‘163 patent are not deemed to be disclosed, inherent, or obvious over Mosberger, then the inclusion of those aspects certainly would be obvious over Mosberger in view of HotLava, under 35 U.S.C. § 103.

It was obvious to supplement the teachings of Mosberger with HotLava because Mosberger expressly invites consideration of a Java-based mechanism for “extending the module graph at runtime.” Specifically, after disclosing that one could “configure a virtual machine module into the graph that would allow interpreted code to be downloaded and executed inside Scout,” Mosberger drops a reference to footnote 39, which directs the reader to a reference entitled “The Java Application Programming Interface.” Ex. 31 at 71, 167. HotLava is precisely the sort of Java-based solution for protocol implementation suggested by Mosberger.

HotLava provides additional disclosure to supplement the teachings of Mosberger on at least the following aspects of the ‘163 patent claims.

i. “A method . . . for processing a message”

Claim 1 recites: “A method in a computer system for processing a message having a sequence of packets, the method comprising” Under Implicit’s apparent claim

constructions, Mosberger in view of HotLava renders obvious this element. *See, e.g.*, Ex. 32 at 93-94, 95, 96, 97.

For example, HotLava states that “Java-based protocols execute alongside other Java applications within a *network computer*” *Id.* at 97. “For each incoming or outgoing *message* a protocol *processes*, it must identify the correct next protocol (if any) and forward the message to it for *processing*.” *Id.* at 94. In the HotLava implementation, a message constitutes a series of “*packets* [that are] are escorted through the protocol graph via non-preemptable threads.” *Id.* at 96.

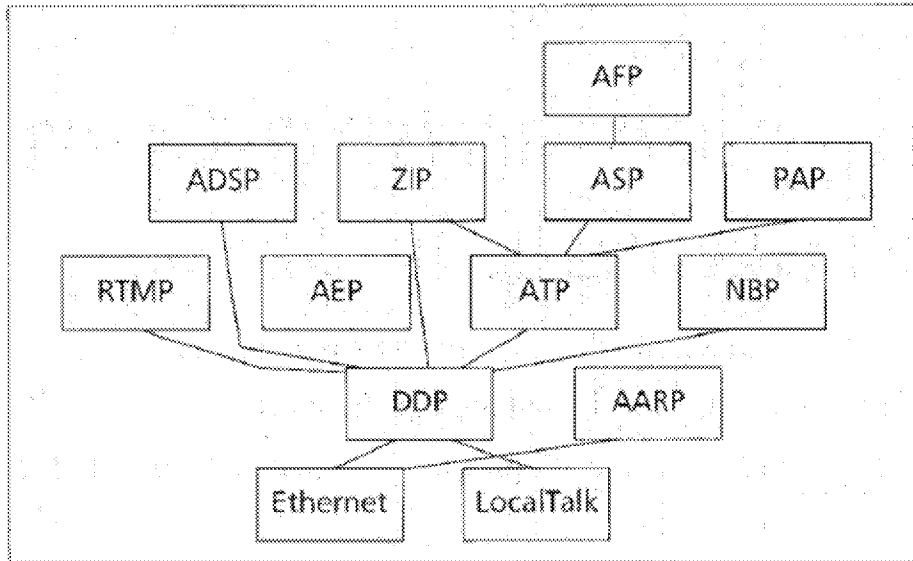
ii. “a plurality of components”

Claim 1 further recites: “providing a plurality of components, each component being a software routine for converting data with an input format into data with an output format.” Under Implicit’s apparent claim constructions, Mosberger in view of HotLava renders obvious this element. *See, e.g.*, Ex. 32 at 93, 94, 95.

HotLava is built to handle components it calls “protocols,” each of which is described as a “*software module* implementing a traditional communication protocol specification (e.g. TCP or IP).” *Id.* The reference goes on to explain:

Protocols are traditionally composed in graphs, which provide services to applications. The arrangements in which protocols (depicted as nodes) may be composed to provide services are described (and in some cases constrained) by a protocol graph (Fig. 1), while a protocol stack is the actual *sequence of protocols* through which messages of a particular session pass. The terms, though, are often used interchangeably.

Id. at 93. Figure 1 of HotLava discloses an example protocol graph illustrating a plurality of components:



■ **Figure 1.** *Example protocol graph (AppleTalk).*

Id. at 94. Moreover, HotLava explains that, “[f]or each *incoming or outgoing message* a protocol processes, it must *identify the correct next protocol* (if any) and forward the message to it for processing.” *Id.*

iii. “*dynamically identifying a non-predefined sequence*”

Claim 1 further recites: “for the first packet of the message, dynamically identifying a non-predefined sequence of components for processing the packets of the message such that the output format of the components of the non-predefined sequence match the input format of the next component in the non-predefined sequence, wherein dynamically identifying includes selecting individual components to create the non-predefined sequence of components after the first packet is received.” Under Implicit’s apparent claim constructions, Mosberger in view of HotLava renders obvious this element. *See, e.g., Ex. 32 at 94, 95, 96, 99.*

HotLava expressly describes itself as a “dynamic” system: “In our Java-based protocol architecture, special service classes *dynamically construct protocol graphs at runtime* as applications need communications services.” *Id.* at 96; *see also id.* at Fig. 1.

HotLava explains that it is “natural to consider” the Java environment as a way of addressing the need for “flexible communication protocols and services to support them,” as a way of solving the problem of “the number and variety of Web- and network-based applications [that] continue[] to increase.” *Id.* at 93. Using the system disclosed in HotLava, “protocols and additional code required to support them can be downloaded and executed on the fly as needed.” *Id.*; *see also id.* at 96 (“This extensible architecture . . . allows on-the-fly introduction of new or replacement protocol code.”). Thus, “new classes, such as those making up our protocol subsystem and protocol implementations, can be added *dynamically* . . .” *Id.* at 95. Among other things, the HotLava approach overcomes shortcomings of certain “traditional” approaches and systems, which had to be “completely recompiled and redeployed” in order to accommodate change. *Id.*

Not only is the protocol graph of software modules (“sequence of protocols”) determined “dynamically . . . at runtime,” each also receives a separate instantiation in memory; thus, “multiple instances of the same protocol can be executing simultaneously.” *Id.* at 98. “For example, an application needing AppleTalk services need only *create an instance* of its corresponding service class.” *Id.* at 96.

As explained earlier, Mosberger proposed configuring “a *virtual machine* module into the graph that would allow interpreted code to be *downloaded and executed* inside Scout.” Ex. 31 at 71. As shown above, HotLava expressly provides “the ability to incorporate new protocol classes . . . into the *virtual machine*.” Ex. 32 at 96. Thus, under the HotLava approach, “protocols and additional code required to support them can be *downloaded and executed* on the fly as needed.” *Id.* at 93. Incorporating into Scout the HotLava approach—a Java-based solution

as expressly proposed in Mosberger—thus clearly satisfies any perceived shortcoming of Mosberger with respect to the “dynamically identifying” limitation.

iv. “*storing an indication of . . . the identified components*”

Claim 1 further recites “storing an indication of each of the identified components so that the non-predefined sequence does not need to be re-identified for subsequent packets of the message.” Under Implicit’s apparent claim constructions, Mosberger in view of HotLava renders obvious this element. *See, e.g., Ex. 32 at 94, 96-97.*

HotJava teaches that “the actual sequence of protocols” (the “protocol stack”) used for a message will pertain to “a particular session pass.” *Id.* at 93. In other words, the system stores “information about what protocol stack should be used *with a given session.*” *Id.* at 94.

The decision to maintain protocol graph configuration as “per-session” information was a conscious design choice in HotLava, to facilitate “flexible communications services”:

The configuration of the protocol graph (e.g., what nodes and arcs it contains) could be placed either in a protocol’s class information or within the state information of each individual instantiation (or protocol object) of that class. Placing the protocol graph in the class fixes that protocol graph configuration for every protocol object instantiation; *placing the protocol graph configuration in the object allows each individual session to determine and dictate its own protocol graph.* Because we desire flexible communications services whereby each application can pick and choose only the functionality it needs, *we chose to place protocol graph configuration in the per-session state information* of protocol objects rather than in their corresponding class.

Id. at 96-97.

v. “*state information*”

Claim 1 finally recites in pertinent part: “for each of a plurality of packets of the message in sequence, for each of a plurality of components in the identified non-predefined sequence, **retrieving** state information relating to performing the processing of the component with the

previous packet of the message; **performing** the processing of the identified component with the packet and the retrieved state information; and **storing** state information relating to the processing of the component with the packet for use when processing the next packet of the message.” Under Implicit’s apparent claim constructions, Mosberger in view of HotLava renders obvious this element. *See, e.g., Ex. 32 at 94, 96-97.*

In a section entitled, “***Maintaining State***,” HotLava discloses a number of details about how state information is handled in the HotLava system. For example, each “software module” (also called a “protocol,” *see id.* at 93) locates relevant “state information” as “one of the first things” that happens “[w]hen a message is received from the network or a user requests that a message be sent.” *Id.* at 94. This can include “per-instance information specific to a particular user session.” *Id.* One specific example of state information provide is “keeping track of time,” which is stored and retrieved and ultimately used to perform steps to “ensur[e] that certain events happen in a timely manner.” *Id.*

(b) *Claim 15*

i. “*demultiplexing packets of messages*”

Claim 15 recites: “A method in a computer system demultiplexing packets of messages.” Under Implicit’s apparent claim constructions, Mosberger discloses this element. *See Claim 1(i)* (showing “A method in a computer system for processing a message having a sequence of packets”) above. Under Implicit’s apparent claim constructions, “demultiplexing” a packet is satisfied by routing a packet to the correct sequence of components for processing it—and Mosberger in combination with HotLava performs this function and renders it obvious. *See Section IV(C) and, e.g., Claim 1(iii)* (showing “dynamically identifying a non-predefined sequence of components for processing the packets of the message”) above; *see also 31 at 94, Fig. 1.*

ii. “dynamically identifying a non-predefined sequence”

Claim 15 further recites “dynamically identifying a non-predefined sequence of components for processing each message based on the first packet of the message so that subsequent packets of the message can be processed without re-identifying the components.”

Under Implicit’s apparent claim constructions, Mosberger in combination with HotLava renders obvious this element. *See* Claim 1(iii) (showing “for the first packet of the message, dynamically identifying a non-predefined sequence of components for processing the packets of the message”) and Claim 1(iv) (showing “storing an indication of each of the identified components so that the non-predefined sequence does not need to be re-identified for subsequent packets of the message”) above.

iii. “different . . . sequences of components can be identified”

Claim 15 further recites “different non-predefined sequences of components can be identified for different messages.” Under Implicit’s apparent claim constructions, Mosberger in combination with HotLava renders obvious this element. *See* 31 at 94, Fig. 1; *see also* Claim 1(ii) and (iii) (showing “plurality of components” and “dynamically identifying includes selecting individual components to create the non-predefined sequence of components”) above.

iv. “each component being a software routine”

Claim 15 further recites “each component being a software routine.” Under Implicit’s apparent claim constructions, Mosberger in combination with HotLava renders obvious this element. *See* Claim 1(ii) (showing “each component being a software routine”) above.

v. “selecting individual components”

Claim 15 further recites “dynamically identifying includes selecting individual components to create the non-predefined sequence of components.” Under Implicit’s apparent claim constructions, Mosberger in combination with HotLava renders obvious this element. *See*

Claim 1(iii) (showing “dynamically identifying includes selecting individual components to create the non-predefined sequence of components”) above.

vi. “state information”

Claim 15 finally recites “for each packet of each message, performing the processing of the identified non-predefined sequence of components of the message wherein state information generated by performing the processing of a component for a packet is available to the component when the component processes the next packet of the message.” Under Implicit’s apparent claim constructions, Mosberger in combination with HotLava renders obvious this element. *See* Claim 1(v) (showing similar “state information” element) above.

(c) Claim 35

i. “instructions for demultiplexing packets of messages”

Claim 35 recites: “A computer-readable medium containing instructions for demultiplexing packets of messages.” Under Implicit’s apparent claim constructions, Mosberger in combination with HotLava renders obvious this element. HotLava is coded in primarily in Java, which can be run using a “virtual machine” on a “variety of platforms without recompilation.” Ex. 32 at 94; *see also id.* at 97 (describing use of native “adapters” such as ones written in “C for Solaris”). One of ordinary skill would have understood that the code for HotLava would be stored and executed on a physical computer-readable medium. *See also* Claim 15(i) showing (“demultiplexing packets of messages”) above.

ii. “dynamically identifying a . . . non-predefined sequence”

Claim 35 recites: “dynamically identifying a message-specific non-predefined sequence of components for processing the packets of each message upon receiving the first packet of the message.” Under Implicit’s apparent claim constructions, Mosberger in combination with HotLava renders obvious this element. *See* Claim 1(iii) (showing “for the first packet of the

message, dynamically identifying a non-predefined sequence of components for processing the packets of the message”) above.

iii. “subsequent packets . . . can use the . . . sequence”

Claim 35 further recites that “subsequent packets of the message can use the message-specific non-predefined sequence identified when the first packet was received.” Under Implicit’s apparent claim constructions, Mosberger in combination with HotLava renders obvious this element. *See* Claim 1(iv) (showing “storing an indication of each of the identified components so that the non-predefined sequence does not need to be re-identified for subsequent packets of the message”) above.

iv. “selecting individual components”

Claim 35 further recites that “dynamically identifying includes selecting individual components to create the message-specific non-predefined sequence of components.” Under Implicit’s apparent claim constructions, Mosberger in combination with HotLava renders obvious this element. *See* Claim 1(iii) (showing “dynamically identifying includes selecting individual components to create the non-predefined sequence of components after the first packet is received”) above.

v. “state information”

Claim 35 further recites “for each packet of the message, invoking the identified non-predefined sequence of components in sequence to perform the processing of each component for the packet wherein each component saves message-specific state information so that that component can use the save message-specific state information when that component performs its processing on the next packet of the message.” Under Implicit’s apparent claim constructions, Mosberger in combination with HotLava renders obvious this element. *See* Claim 1(v) (showing similar “state information” element) above.

4. HotLava Anticipates Claims 1, 15, and 35 Under § 102(a) and (b)

The HotLava reference not only renders the claims of the '163 patent when considered in combination with Mosberger, but HotLava also independently and standing alone discloses each and every element of claims 1, 15, and 35. Accordingly, HotLava also fully anticipates these claims for the reasons set forth in detail above, which are incorporated by reference in this proposed ground of rejection.

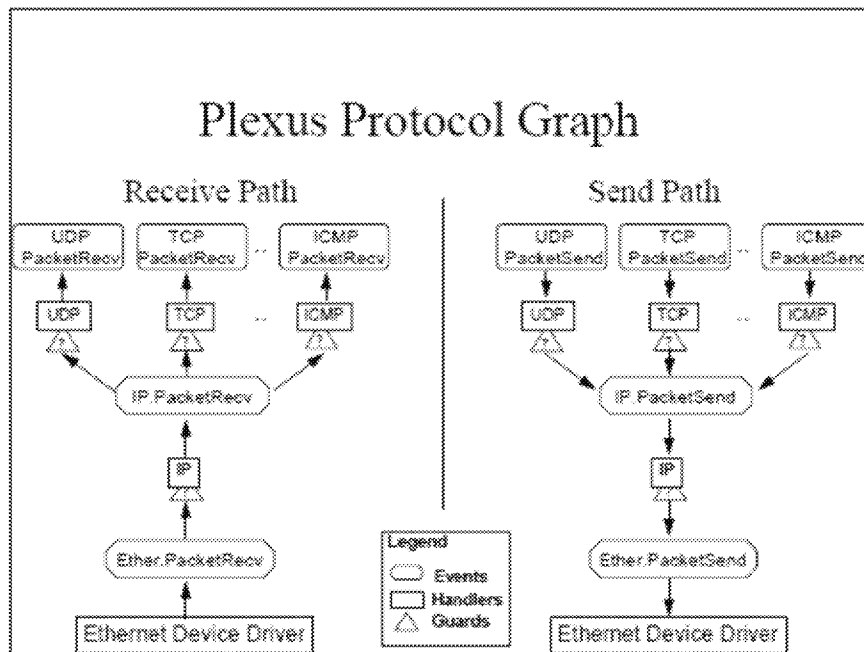
5. Mosberger in View of Plexus Renders Obvious Claims 1, 15, and 35 Under § 103

The article “An Extensible Protocol Architecture for Application-Specific Networking” (Ex. 33, “Plexus”) by Marc Fiuczynski *et. al* was published in 1996 in Proceedings of the 1996 Winter USENIX Conference. It describes a system called “Plexus,” which is a “networking architecture that allows applications to achieve high performance with customized protocols.” Plexus was not considered during the prosecution of the '163 patent.

If certain aspects recited in claims 1, 15, and 35 of the '163 patent are not deemed to be disclosed, inherent, or obvious over Mosberger, then the inclusion of those aspects certainly would be obvious over Mosberger in view of Plexus, under 35 U.S.C. § 103.

It was obvious to supplement the teachings of Mosberger with Plexus because Mosberger expressly states it is “*straight-forward to add a dynamic module-loading facility* to Scout” (Ex. 31 at 71), and a “key aspect of Plexus is . . . [a] protocol graph that can be *dynamically* changed as applications come and go.” Ex. 33 at 55. Plexus does so in a way that “does not compromise the safety of other applications or the operating system.” *Id.* at 55.

Following is an illustration of the manner in which Plexus performs what it expressly calls “demultiplexing” in a protocol stack:



Id. at 57, Fig. 1. “Packets sent by the application are pushed down the graph until they reach the actual device.” *Id.*

“Plexus allows applications to define new protocols or to change the implementation of existing protocols.” *Id.* Indeed, Plexus even “supports multiple implementations of the same protocol for different endpoints.” *Id.* at 58.

The Plexus system is also “dynamic” under Implicit’s apparent claim construction because it permits “[r]untime adaptation.” *Id.* at 56. Specifically, “[a]pplications may add extensions to the kernel at any point during the system’s execution without requiring superuser privileges or a system reboot.” *Id.*; *see also id.* (“Plexus allows extensions to be safely loaded and unloaded into a running system . . .”).

Thus, to the extent that Mosberger is deemed to lack inadequate disclosure of the “dynamically identifying” limitation for claims 1, 15, and 35, the combination of Mosberger with Plexus clearly makes up for any such perceived deficiency.

6. **Mosberger in View of ComScript Renders Obvious Claims 1, 15, and 35 Under § 103**

The article “ComScript: An Environment for the Implementation of Protocol Stacks and their Dynamic Reconfiguration” (Ex. 34, “ComScript”) by Murhimanya Muhugusa *et. al* was published in December 1994. It describes a communication protocol implementation called “ComScript.” ComScript was not considered during the prosecution of the ‘163 patent.

If certain aspects recited in claims 1, 15, and 35 of the ‘163 patent are not deemed to be disclosed, inherent, or obvious over Mosberger, then the inclusion of those aspects certainly would be obvious over Mosberger in view of ComScript, under 35 U.S.C. § 103.

It was obvious to supplement the teachings of Mosberger with ComScript because Mosberger expressly states it is “*straight-forward to add a dynamic module-loading facility to Scout*” (Ex. 31 at 71), and Plexus expressly proposes an approach that “brings more flexibility by allowing an application to dynamically (re)configure an entire protocol stack” Ex. 34 at 1.

ComScript provides, as an illustration, the following example of how the disclosed system can be used to create a new protocol stack or sequence of “modules” on the fly for purposes of a given session between hosts A and B:

An application running on host A establishes a communication with a COMSCRIPT server (CS) on the remote machine B by opening two connections, one for control information and the other for data exchange. The control connection is used by the application to send requests to the remote server. The application then *downloads* its own code to host B using the control channel. The *execution* of this code in the remote host results in the *creation of a protocol stack* which can then be used by the application to exchange data with host B.

Id. at 6-7; Fig. 10; see also Figs. 7-9 (illustrating how to add or remove a “module” from a stack; “the number of configurable entities is unlimited”).

The ComScript system is also “dynamic” under Implicit’s apparent claim construction because it expressly states that one its “primary goal[s]” is to “make protocol stacks truly *configurable at run time.*” *Id.* at 8.

Thus, to the extent that Mosberger is deemed to lack inadequate disclosure of the “dynamically identifying” limitation for claims 1, 15, and 35, the combination of Mosberger with ComScript clearly makes up for any such perceived deficiency.

VI. CERTIFICATION PURSUANT TO 37 C.F.R. § 1.915(B)(7)

The Requester hereby certifies that the estoppel provisions of 37 C.F.R. § 1.915(b)(7) would not prohibit the granting of this Request for *Inter Partes* Reexamination.

VII. IDENTIFICATION OF REAL PARTY IN INTEREST PURSUANT TO 37 C.F.R. § 1.915(B)(8)

The real party in interest is Juniper Networks, Inc.

VIII. CONCLUSION

The Requester requests that claims 1, 15, and 35 of the ‘163 patent be reexamined in view of the prior art and grounds discussed in this Request for *Inter Partes* Reexamination. The Requester also requests the issuance of a certificate under 35 U.S.C. § 316(a) cancelling at least claims 1, 15, and 35.

Respectfully submitted,
IRELL & MANELLA LLP



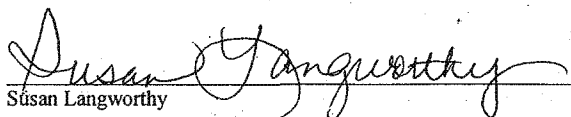
David McPhie, Reg. No.56,412

Dated: February 13, 2012

840 Newport Center Drive, Suite 400
Newport Beach, CA 92660
(949)760-0991

Certificate of Mailing

I hereby certify that this correspondence is being deposited with the U.S. Postal Service as Express Mail Label Nos. EM305148562US and EM305148576US addressed to: Mail Stop *Inter Partes* Reexam, Central Reexamination Unit, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on February 13, 2012.



Susan Langworthy