

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

| | | | |
|-------------------|--|----------------------|------------------|
| In re Patent No.: | 7,711,857 | Group Art Unit: | To be assigned |
| Inventors: | Edward Balassanian | Examiner: | To be assigned |
| Issued: | May 10, 2010 | Attorney Docket No.: | 159291-0025(163) |
| Serial No.: | 11/933,022 | Reexam Control No.: | To be assigned |
| Title: | METHOD AND SYSTEM FOR DATA DEMULTIPLEXING | 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. 7,711,857 ("the '857 patent") entitled "Method and System for Data Demultiplexing." 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 '857 patent was filed on October 31, 2007 and issued on May 4, 2010. The patent has not yet expired. Implicit Networks, Inc. ("Implicit") has alleged that it is the current assignee of the '857 patent. A copy of the '857 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 '857 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 | 2 |
| II. DISCLOSURE OF CONCURRENT PROCEEDINGS | 11 |
| III. CLAIMS FOR WHICH REEXAMINATION IS REQUESTED AND CITATION OF PRIOR ART | 12 |
| IV. CLAIM CONSTRUCTION ADMISSIONS OF THE PATENT OWNER | 21 |
| V. PERTINENCE AND MANNER OF APPLYING THE PRIOR ART | 28 |
| VI. CERTIFICATION PURSUANT TO 37 C.F.R. § 1.915(b)(7) | 286 |
| VII. IDENTIFICATION OF REAL PARTY IN INTEREST PURSUANT TO 37 C.F.R. § 1.915(b)(8) | 286 |
| VIII. CONCLUSION | 286 |

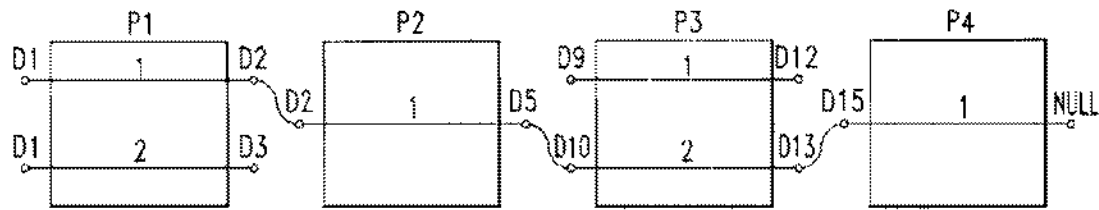
I. INTRODUCTION

The PTO should grant this Request and initiate *inter partes* reexamination proceedings for the '857 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 '857 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 ‘857 patent claims priority to a patent application filed on December 29, 1999 (App. No. 09/474,664) which issued as U.S. Pat. No. 6,629,163 (“the ‘163 patent”) on September 30, 2003. *See* Ex. 49 (‘163 patent). A request for *ex parte* reexamination of the ‘163 patent was submitted on December 15, 2008. The ensuing reexamination proceeding (“the ‘163 Reexamination”) resulted in issuance of an Ex Parte Reexamination Certificate (7567th) on June 22, 2010. *See* Ex. 2 (‘163 Reexamination Certificate). The ‘857 patent has not yet been subject to either *ex parte* or *inter partes* reexamination.

The ‘857 patent describes itself as relating “generally to a computer system for data demultiplexing.” Ex. 1 (‘857 patent) at 1:13-14, 2:58-65. As explained in the background section of the ‘857 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 1:21-28. 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 2:39-41. 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 2:48-50. An illustration of such a conversion (from format D1 to D15) is illustrated in Figure 2 of the ‘857 patent:



During the original prosecution and prior *ex parte* reexamination proceedings for the '163 patent (parent to the '857 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

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

amended. The allowance was expressly based on the patentee's argument that "Mosberger does not dynamically identify sequences."

The prosecution of the '857 patent included *none* of this scrutiny and discussion of the Mosberger reference, even though the time period of this prosecution (from October 31, 2007 to May 4, 2010) largely overlapped with the '163 reexamination proceeding (from December 15, 2008 to June 22, 2010). Indeed, even though Mosberger was presented in the '163 reexamination request of December 15, 2008, was found to present a substantial new question of patentability against the '163 claims on January 17, 2009, and was the sole basis for the rejection of all '163 claims in an office action dated July 7, 2009, the patentee did not submit an Information Disclosure Statement regarding Mosberger in the '857 patent prosecution until January 29, 2010—several weeks *after* the '857 patent examiner had declared that the pertinent objections over the prior art had been overcome. *See* Ex. 40-F ('857 1/29/2010 IDS) at 3; Ex. 40-D ('857 12/11/2009 Final Rejection) at 3 ("Claims 6, 8, 9, 22-24 and 26-28 would be allowable if a terminal disclaimer is filed to overcome the obviousness-type double patenting rejection.").

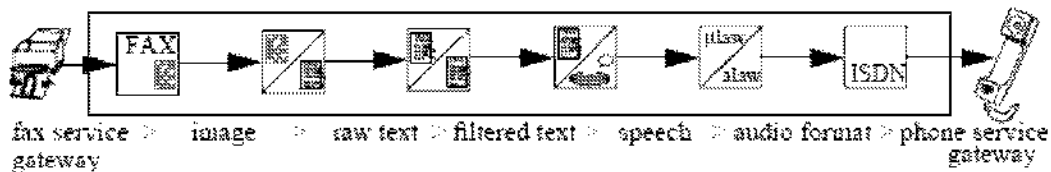
Thus, the examiner for the '857 patent prosecution was not made aware of the simultaneously ongoing battle regarding the patentability of the '163 patent claims over Mosberger. The patentee failed to timely bring these issues to the attention of the examiner in the '857 patent prosecution, including, in particular, a complete omission of the patentee's office action response in the '163 patent reexamination dated February 8, 2010 (Ex. 35-M)—a document that Implicit has now characterized as "*the most important document in the entire case*" against Requester in the concurrent district court proceedings. Ex. 37-C (Implicit's Reply of December 19, 2011 to Defendants'

Responsive Claim Construction Brief) at 2. Accordingly, Requester now submits these materials and issues so that the PTO may properly consider them in the first instance as they bear upon the invalidity of the claims of the '857 patent.

Because Mosberger was not timely disclosed during prosecution of the '857 patent, prosecution focused entirely on another prior art reference known as Taylor (U.S. Patent No. 6,785,730). The most prominent deficiency of Taylor alleged by patentee was that Taylor purportedly discloses only a single component, and several of the amendments made during prosecution related to this deficiency. *See* Ex. 40-C (9/24/2009 Amendment) at 5-7 (arguing that Taylor fails to disclose elements “relating to a *plurality* of components in a sequence”). The patentee also alleged that Taylor failed to provide adequate detail on the internal operation of its format translator component. *Id.* at 5-7 (“Taylor provides no details on the internal operation of the format translator 32.”). *Id.* at 5-7, 12 (independent claim 6), 14 (independent claim 22). The examiner ultimately allowed these claims without further explanation. Ex. 40-G (3/10/2010 Allowance).

All of the possible points of distinction raised during prosecution of the '857 patent (and its parent, the '163 patent) are now clearly and repeatedly addressed by the prior art presented in this Request.

For example, a technical paper presented at an international telecommunications conference in 1996 (“Pfeifer96”) demonstrates that researchers had already discovered how to process a message using a “dynamically generated converter *chain*” consisting of *multiple components*. Ex. 3 at 124 (emphasis added), 109-11.

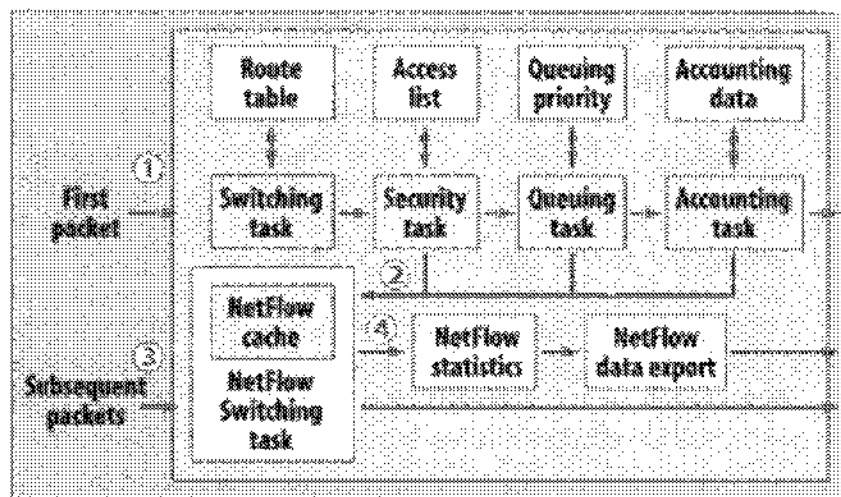


Id. at 111 (Figure 6: “Converter chain: fax reception, conversion to text and speech, telephone delivery”). Pfeifer96 also provides internal details regarding the operation of its processing steps, observing, *e.g.*, that selection of the chain of components is based on analysis of a data type of the incoming message (*e.g.*, whether its source “medium” is fax, voice call, multimedia email, and so on) and of the message’s intended destination (called party). *Id.* at 120, 109-11. This reference was published *over three years* before the filing date of the application that became the parent of the ‘857 patent. Pfeifer96 fully anticipates and renders obvious every element of claims 1, 4, and 10 of the ‘857 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 ‘857 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 *several* distinct functions are applied task-by-task by components to the packets of a particular flow. *E.g.*, Ex. 15 at 4:20-59. The various functions to be applied to a flow are ascertained while processing the first packet of the flow, and this information is then “cache[d]” for high-speed use by subsequent packets. Kerr also provides internal details regarding its classification and treatment of flows, including

analysis of a data type (e.g., “standard port numbers includ[ing] . . . FTP . . . TELNET . . . an internet telephone protocol, or an internet video protocol”), and analysis of multiple packet headers (including, e.g., “source” and “destination . . . IP . . . address” at layer 3, and “port number[s]” at layer 4). *Id.* at 3:3-20. 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, 4, and 10 of the '857 patent, both on its own and in combinations with other references as set forth in this Request.

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

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 multiple “code modules, called *plugins*, to be dynamically added and configured at runtime.” Ex. 25 (Decasper98) at 1. A sequence of *multiple plugin components* may be assigned to process the packets of each flow. *E.g., id.* at 5 (“Each flow table entry stores pointers [*plural*] to the appropriate plugins [*plural*]”). 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 3. Decasper98 also provides internal details regarding its classification and treatment of flows, including analysis of a data type (*e.g.*, “protocol” and “port”), and analysis of multiple packet headers (including, *e.g.*, “source” and “destination address” at layer 3, and “source” and “port” at layer 4). *Id.* at 3.

As mentioned above, this Request also presents—for the first time—a robust analysis and application of the Mosberger reference to the claims of the ‘857 patent. The Request further explains how Mosberger is not nearly so limited as the patentee argued to the PTO during the prior *ex parte* reexamination proceedings for the parent of the ‘857 patent. Mosberger itself states that it would be “*straight-forward* to add a *dynamic module-loading facility*.” Ex. 31 (Mosberger) at 71. And to the extent that Mosberger is still deemed to lack any elements of the ‘857 patent claims, obviousness combinations are presented with related publications by others (*e.g.*, the “Plexus” reference), at least one of which also anticipates the ‘857 patent claims standing alone (the “HotLava” reference).

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 claims 1, 4, and 10 of the ‘857 patent, and a certificate under 35 U.S.C. § 316(a) ultimately issued cancelling all of these claims.

II. DISCLOSURE OF CONCURRENT PROCEEDINGS

A request for inter partes reexamination is already pending for the ‘163 patent, which is the parent of the ‘857 patent. Requester filed this earlier request on February 13, 2012, and it has now been assigned control number 95/000,659.

Implicit has asserted the ‘857 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 ‘857 patent by assignment. Implicit alleges that claims 1, 4, and 10 of the ‘857 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 (Infringement Contentions), 36-E and 36-F ('857 claim charts).

A Markman hearing was held on January 18-19, 2012, and the District Court recently issued a Claim Construction Order on February 29, 2012. *See* Ex. 39 (Claim Construction Order).

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

Reexamination of claims 1, 4, and 10 of the '857 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 ³ |
|---|-------------------|----------------------|
| 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 |

³ For the convenience of the examiner and the parties, exhibit numbers have largely been maintained across the *inter partes* reexamination requests for the '163 and '857 patents.

| Prior Art Reference | Prior Art Date | Exhibit³ |
|--|-----------------------|----------------------------|
| 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 |
| Specification entitled RFC 793: "Transmission Control Protocol" by Information Sciences Institute ("RFC 793") | September 1981 | Ex. 9 |
| Thesis entitled "Generic Description of Telecommunication Services and Dynamic Resource Selection in Intelligent Communication Environments" by Stefan Arbanowski ("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 |

| Prior Art Reference | Prior Art Date | Exhibit ³ |
|--|--------------------|----------------------|
| 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 <i>et al.</i> ("Bellare97") | October 27, 1997 | Ex. 17 |
| 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 <i>et al.</i> ("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 <i>et al.</i> ("Dietz") | June 30, 1999 | Ex. 22 |
| Article entitled "Dynamic Reconfiguration of Agent-Based Applications") by Luc Bellissard, Noel de Palma, and Michel Riveill ("Bellissard") | September 10, 1998 | Ex. 23 |
| Publication entitled "DTE Firewalls Phase Two Measurement and Evaluation Report" by Timothy L. Fraser <i>et al.</i> of Trusted Information Systems ("Fraser") | July 22, 1997 | Ex. 24 |
| Article entitled "Router Plugins: A Software Architecture for Next Generation Routers" by Dan Decasper <i>et al.</i> ("Decasper98") | September 1998 | Ex. 25 |

| Prior Art Reference | Prior Art Date | Exhibit ³ |
|---|--------------------|----------------------|
| 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 <i>et al.</i> ("RFC 1829") | August 1995 | Ex. 27 |
| Specification entitled RFC 1883: "Internet Protocol, Version 6 (IPv6) Specification" by S. Deering and R. Hinden ("RFC 1883") | December 1995 | Ex. 28 |
| Article entitled "Crossbow: A Toolkit for Integrated Services over Cell Switched IPv6" by Dan Decasper <i>et al.</i> ("Decasper97") | May 28, 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 |
| Article entitled "The Active IP Option" by David J. Wetherall <i>et al.</i> ("Wetherall") | September 11, 1996 | Ex. 47 |
| Paper entitled "Active Gateway: A Facility for Video Conferencing Traffic Control" by Shunge Li <i>et al.</i> ("Li") | February 1, 1997 | Ex. 48 |

Most of these prior art references were not cited or considered by the PTO during prosecution of the '857 patent and are not cumulative to the art of record in the original file. Only two of the references relied upon in this Request were cited during the prosecution of the '857 patent (*i.e.*, Mosberger and Dietz). However, neither was discussed or applied by the Examiner at any time during prosecution of the '857 patent, and 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 bears the date November 27, 1996, and is prior art under 35 U.S.C. § 102(a) and (b). Pfeifer96a is substantively identical to Pfeifer96, except its figures are rendered in color. Pfeifer96a also bears the date November 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. 43 (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. 44 (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).

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

Pfeifer97 bears the date December 19, 1997, 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 date October 22, 1997, and 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).

Bellissard was published on September 10, 1998. *See* Ex. 45 (document showing date of publication). 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 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. 46 (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 bears the date May 28, 1997, and 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)

Wetherall bears the date September 11, 1996, and is prior art under 35 U.S.C. § 102(a) and (b).

Li bears the date February 1, 1997, and is prior art under 35 U.S.C. § 102(a) and (b).

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 |
|---|-----------------|
| '163 patent | Ex. 49 |
| '163 patent: Ex Parte Reexamination Certificate | Ex. 2 |
| '163 FH: Original Claims | Ex. 35-A |
| '163 FH: 9/23/2002 Office Action | Ex. 35-B |
| '163 FH: 2/24/2003 Amendment | Ex. 35-C |
| '163 FH: 5/20/2003 Notice of Allowance | Ex. 35-D |
| '163 Reexam FH: Ex Parte Reexamination Request | Ex. 35-E |
| '163 Reexam FH: Order Granting Ex Parte Reexamination Request | Ex. 35-F |
| '163 Reexam FH: 7/7/2009 Office Action | Ex. 35-G |
| '163 Reexam FH: 9/1/2009 Amendment | Ex. 35-H |
| '163 Reexam FH: 10/23/2009 Interview Summary | Ex. 35-I |

| Other Written Evidence | Exhibit |
|---|-----------------|
| '163 Reexam FH: 12/4/2009 Final Office Action | Ex. 35-J |
| '163 Reexam FH: 12/18/2009 Response to Final Rejection | Ex. 35-K |
| '163 Reexam FH: 1/21/2010 Advisory Action | Ex. 35-L |
| '163 Reexam FH: 2/4/2010 Examiner Interview Summary | Ex. 35-O |
| '163 Reexam FH: 2/8/2010 Amendment After Final | Ex. 35-M |
| '163 Reexam FH: 3/2/2010 Notice of Intent to Issue Certificate | Ex. 35-N |
| '857 FH: Original Claims | Ex. 40-A |
| '857 FH: 2/19/2008 Office Action | Ex. 40-B |
| '857 FH: 6/24/2009 Amendment | Ex. 40-C |
| '857 FH: 12/11/2009 Final Rejection | Ex. 40-D |
| '857 FH: 1/29/2010 Amendment | Ex. 40-E |
| '857 FH: 1/29/2010 IDS | Ex. 40-F |
| '857 FH: 3/10/2010 Notice of Allowance | Ex. 40-G |
| First Amended Complaint | Ex. 36-A |
| Implicit Patent Infringement Contentions | Ex. 36-B |
| Implicit '163 Infringement Claim Chart (Security Devices) | Ex. 36-C |
| Implicit '163 Infringement Claim Chart (Application Acceleration) | Ex. 36-D |
| Implicit '857 Infringement Claim Chart (Security Devices) | Ex. 36-E |
| Implicit '857 Infringement Claim Chart (Application Acceleration) | Ex. 36-F |
| Implicit Opening Claim Construction Brief | Ex. 37-A |
| 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 |

| Other Written Evidence | Exhibit |
|--|---------|
| Claim Construction Order | Ex. 39 |
| Specification entitled RFC 791: “Internet Protocol: DARPA Internet Program Protocol Specification” by Information Sciences Institute in September 1981 (“RFC 791”) | Ex. 41 |
| Specification entitled RFC 1700: “Assigned Numbers” by Network Working Group in October 1994 (“RFC 1700”) | Ex. 42 |
| Book entitled “IPv6: The New Internet Protocol” by Christian Huitema (“Huitema”) published October 28, 1997 | Ex. 29 |
| Document showing publication date of Comscript (Penn State University, CiteSeer Digital Library) | Ex. 38 |
| Document showing publication date of Nelson | Ex. 43 |
| Document showing publication date of Cox | Ex. 44 |
| Document showing publication date of Bellissard | Ex. 45 |
| Document showing publication date of Huitema | Ex. 46 |
| Deposition of David Mosberger, Ph.D., September 16, 2011 | Ex. 50 |

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 ‘857 patent, 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. The District Court in the concurrent litigation proceedings issued an order two days ago construing certain claim terms of the ‘857 patent. *See* Ex. 39. However, the 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.⁴

A. Original Prosecution of the ‘163 Patent

During the original prosecution of the ‘163 patent (the parent of the ‘857 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.*

⁴ Moreover, nothing in this Request should be construed as expressing any position as to whether the claims of the ‘857 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 of the '163 Patent

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 (7/07/2009 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 (9/01/2009 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 (3/02/2010 Notice of Intent to Issue Ex Parte Reexamination Certificate) at 4.

C. Prosecution of the ‘857 Patent

During the original prosecution of the ‘857 patent, the patentee initially proposed 25 claims. Ex. 40-A (Original Claims) at 21-25. All of these claims were rejected over a single prior art reference known as Taylor (U.S. Patent No. 6,785,730). Ex. 40-B (6/24/2009 Initial Office Action) at 3-9. In order to distinguish Taylor, the patentee amended the claims to add several new limitations. Ex. 40-C (9/24/2009 Amendment) at 4-8.

The most prominent deficiency of Taylor alleged by patentee was that Taylor discloses only a single component, and several of the newly added limitations relate to this deficiency—as argued by the patentee:

(1) “a *sequence* of components”—arguing “Taylor only discloses a single format translator 32 and thus cannot teach or suggest storing state information relating to a *plurality* of components in a sequence.” *Id.* at 5-7, 12 (independent claim 6), 14 (independent claim 22).

(2) “storing state information for each of a *plurality* of components”—arguing “Taylor only discloses a single format translator 32 and thus cannot teach or suggest storing state information relating to a *plurality* of components in a sequence.” *Id.*

(3) “storing an indication of the identified *components*”—arguing: “Given that Taylor only has a single component for the format translation, there is no suggestion of storing an indication for each of the identified components.” *Id.*

A second deficiency of Taylor alleged by patentee was that Taylor provides no internal details on the internal operation of its format translator component, and two of the newly added limitations relate to this deficiency:

(1) “*analyz[ing] the data type* of a first packet”—arguing “Taylor provides no details on the internal operation of the format translator 32.” *Id.* at 5-7, 12 (independent claim 6), 14 (independent claim 22).

(2) “*analyzing the plurality of headers* of a first packet”—arguing “Taylor provides no details on the internal operation of the format translator 32.” *Id.*

A third deficiency of Taylor which the patentee attempted to argue was that Taylor “only deals with one message at a time,” but all of the claims containing newly added limitations relating to this alleged deficiency (*e.g.*, “a *plurality* of messages”) were subsequently rejected by the Examiner and abandoned by the patentee. *Id.* at 11-13 (claims 1, 10, 19); Ex. 40-D (12/11/2009 Final Rejection) at 4-8; Ex. 40-E (1/29/2010 Amendment) at 2-4.

Apparently persuaded regarding the first two deficiencies of Taylor but without explaining the reasoning behind the decision, the Examiner indicated that the claims containing the newly added limitations relating to those two deficiencies would be allowable (*i.e.*, independent claims 6 and 22 and their dependent claims). Ex. 40-D (12/11/2009 Final Rejection) at 3 (“Claims 6, 8, 9, 22-24 and 26-28 would be allowable if a terminal disclaimer is filed to overcome the obviousness-type double patenting rejection”).

The patentee canceled the remaining claims, and added a single new independent claim parallel to allowable method claim 6⁵ but instead reciting a “computer-readable storage medium.” Ex. 40-E (1/29/2009 Amendment) at 2-4. And thus the three independent claims reciting limitations relating to the first two deficiencies of Taylor were allowed (claims 6, 22, and 29), as were their dependent claims. Ex. 40-G (3/10/2010 Notice of Allowance) at 1 (without explaining reasoning behind decision).

D. Admissions Regarding Claim Construction

In addition to statements made during the prosecution history of the ‘857 and ‘163 patents, 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 ‘857 patent. *See* Exs. 36-A (Complaint), 36-B (Infringement Contentions), 36-C and 36-D (‘163 claim charts), 36-E and 36-F (‘857 claim charts). 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

⁵ The amendment inaccurately asserts “newly added claim 29 . . . parallel[s] the structure of method claim 22.” *See* Ex. 40-E (1/29/2009 Amendment) at 5, 2-4.

a technical tutorial describing the purported scope of the '857 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 |
|--|---|
| 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 '857 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 2:62-65; *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 ‘857 and ‘163 patents:

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, 4, and 10 of the ‘857 patent 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:

- Decasper98 and its obviousness combinations begin on page 33.
- Mosberger and its obviousness combinations begin on page 100.
- HotLava begins on page 126.
- Pfeifer96 and its obviousness combinations begin on page 130.
- Kerr and its obviousness combinations begin on page 224.

The following table summarizes the reasons for reexamination being sought, along with page numbers for each corresponding section in this Request.

| Claims | Grounds of Unpatentability | Statutory Basis | Page |
|---------------|---|------------------------|-------------|
| 1, 4, 10 | Anticipated by Decasper98 | 102(a),(b) | |
| 1, 4, 10 | Obvious over Decasper98 | 103 | |
| 1, 4, 10 | Obvious over Decasper98 in view of RFC 1825 and RFC 1829 | 103 | |
| 1, 4, 10 | Obvious over Decasper98 in view of RFC 1883 | 103 | |
| 1, 4, 10 | Obvious over Decasper98 in view of Decasper97 | 103 | |
| 1, 4, 10 | Obvious over Decasper98 in view of Decasper97, Bellare97, and Bellare95 | 103 | |
| 1, 4, 10 | Obvious over Decasper98 in view of IBM96 | 103 | |

| | | |
|----------|---|------------|
| 1, 4, 10 | Obvious over Decasper98 in view of IBM96 and Nelson | 103 |
| 1, 4, 10 | Obvious over Decasper98 in view of RFC 1825, RFC 1829, Decasper97, Bellare97, Bellare95, IBM96, and Nelson | 103 |
| 1, 4, 10 | Obvious over Decasper98 in view of Fraser | 103 |
| 1, 4, 10 | Obvious over Decasper98 in view of Fraser, RFC 1825, and RFC 1829 | 103 |
| 1, 4, 10 | Obvious over Decasper98 in view of Bellissard | 103 |
| 1, 4, 10 | Obvious over Decasper98 in view of Bellissard, RFC 1825, and RFC 1829 | 103 |
| 1, 4, 10 | Obvious over Decasper98 in view of Wetherall | 103 |
| 1, 4, 10 | Obvious over Decasper98 in view of Wetherall, RFC 1825, and RFC 1829 | 103 |
| 1, 4, 10 | Obvious over Decasper98 in view of RFC 1825, RFC 1829, RFC 1883, Decasper97, Bellare97, Bellare95, IBM96, Nelson, Fraser, Bellissard, and Wetherall | 103 |
| 1, 4, 10 | Anticipated by Mosberger | 102(a),(b) |
| 1, 4, 10 | Obvious over Mosberger | 103 |
| 1, 4, 10 | Obvious over Mosberger in view of HotLava | 103 |
| 1, 4, 10 | Anticipated by HotLava | 102(a),(b) |
| 1, 4, 10 | Obvious over Mosberger in view of Plexus | 103 |
| 1, 4, 10 | Obvious over Mosberger in view of Comscript | 103 |
| 1, 4, 10 | Anticipated by Pfeifer96 | 102(a),(b) |
| 1, 4, 10 | Obvious over Pfeifer96 | 103 |

| | | |
|----------|---|-----|
| 1, 4, 10 | Obvious over Pfeifer96 in view of ISDN98 and Nelson | 103 |
| 1, 4, 10 | Obvious over Pfeifer96 in view of Arbanowski96 | 103 |
| 1, 4, 10 | Obvious over Pfeifer96 in view of Pfeifer97 | 103 |
| 1, 4, 10 | Obvious over Pfeifer96 in view of Cox | |
| 1, 4, 10 | Obvious over Pfeifer96 in view of Meer96 | 103 |
| 1, 4, 10 | Obvious over Pfeifer96 in view of Meer96 and RFC 793 | 103 |
| 1, 4, 10 | Obvious over Pfeifer96 in view of Franz98 | 103 |
| 1, 4, 10 | Obvious over Pfeifer96 in view of ISDN98, Nelson, Cox, Meer96, RFC 793, and Franz98 | 103 |
| 1, 4, 10 | Obvious over Pfeifer96 in view of Wetherall | 103 |
| 1, 4, 10 | Obvious over Pfeifer96 in view of Wetherall, ISDN98, and Nelson | 103 |
| 1, 4, 10 | Obvious over Pfeifer96 in view of Li | 103 |
| 1, 4, 10 | Obvious over Pfeifer96 in view of Li, ISDN98, and Nelson | 103 |
| 1, 4, 10 | Obvious over Pfeifer96 in view of Wetherall and Li | 103 |
| 1, 4, 10 | Obvious over Pfeifer96 in view of Wetherall, Li, ISDN98, and Nelson | 103 |
| 1, 4, 10 | Obvious over Pfeifer96 in view of Pfeifer97 and Alam | 103 |
| 1, 4, 10 | Obvious over Pfeifer96 in view of Pfeifer97 and Yun | 103 |
| 1, 4, 10 | Obvious over Pfeifer96 in view of Meer96, Arbanowski96, Pfeifer97, and Franz98 | 103 |

| | | |
|----------|---|--------|
| 1, 4, 10 | Obvious over Pfeifer96 in view of Meer96, Arbanowski96, Pfeifer97, Franz98, ISDN98, Nelson, Cox, RFC 793, Alam, and Yun | 103 |
| 1, 4, 10 | Anticipated by Kerr | 102(e) |
| 1, 4, 10 | Obvious over Kerr | 103 |
| 1, 4, 10 | Obvious over Kerr in view of NetFlow | 103 |
| 1, 4, 10 | Obvious over Kerr in view of RFC 1825 and 1829 | 103 |
| 1, 4, 10 | Obvious over Kerr in view of Bellare97 and Bellare95 | 103 |
| 1, 4, 10 | Obvious over Kerr in view of IBM96 | 103 |
| 1, 4, 10 | Obvious over Kerr in view of IBM96 and Nelson | 103 |
| 1, 4, 10 | Obvious over Kerr in view of RFC 1825, RFC 1829, Bellare97, Bellare95, IBM96, and Nelson | 103 |
| 1, 4, 10 | Obvious over Kerr in view of Fraser | 103 |
| | Obvious over Kerr in view of Fraser in view of Bellare97 and Bellare95 | 103 |
| 1, 4, 10 | Obvious over Kerr in view of Bellissard | 103 |
| 1, 4, 10 | Obvious over Kerr in view of Bellissard in view of Bellare97 and Bellare95 | 103 |
| 1, 4, 10 | Obvious over Kerr in view of Wetherall | 103 |
| 1, 4, 10 | Obvious over Kerr in view of Wetherall in view of Bellare97 and Bellare95 | 103 |
| 1, 4, 10 | Obvious over Kerr in view of RFC 1825, RFC 1829, Bellare97, Bellare95, IBM96, Nelson, Fraser, Bellissard, and Wetherall | 103 |
| 1, 4, 10 | Obvious over Kerr in view of Checkpoint and Shwed | 103 |

| | | |
|----------|--|-----|
| 1, 4, 10 | Obvious over Kerr in view of Dietz | 103 |
| 1, 4, 10 | Obvious over Kerr in view of Pfeifer96 | 103 |
| 1, 4, 10 | Obvious over Kerr in view of Pfeifer96, ISDN98, and Nelson | 103 |

A. Decasper98 (Exhibit 25)

The article “Router Plugins: A Software Architecture for Next Generation Routers” by Dan Decasper *et al.* (“Decasper98”) was published in September 1998, and it was not considered during prosecution of the ‘857 patent.

1. Decasper98 Anticipates Claims 1, 4, and 10 Under § 102(a), (b)

(a) Claim 1

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

Claim 1 recites: “A method in a computer system for processing packets of a message, the method comprising” Under Implicit’s apparent claim constructions, Decasper98 discloses this element.

Claim 1 recites a method in “a computer system.” 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 recites the method is for processing “packets of a message.” 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 (emphasis added). A flow is defined as the set of packets which share the same values for the following six header fields: “*<source address, destination address, protocol, source port, destination port, incoming interface>*.” *Id.* at 3 (“Filters are specified as six-tuples: *source address, destination address, protocol, source port, destination port, incoming interface>*”), 5 (“entries in the flow table are identified by the same six tuple used to specify filters, but without masks or wildcards (all fields have fully specified values). In other words, a flow table entry unambiguously identifies a particular flow.”), 9 (“each entry in the flow table corresponds to a flow with a fully specified filter (one that contains no wildcards)”). A flow would comprise a “message” under Implicit’s apparent claim constructions. *See* Section IV.

Claim 1 recites the method is for “processing” packets of a message. “One of the novel features” of the Decasper98 design “is the ability to bind different plugins to individual flows.” *Id.* at 1. The various plugins which may be bound to individual flows include, *e.g.*, “plugins for IP security” (*e.g.*, “authentication and/or encryption”), “plugins implementing IPv6 options,” “plugins for packet scheduling,” and a “statistics gathering plugin.” *Id.* at 4-5. Under Implicit’s apparent claim constructions, the various operations performed by these plugins on the packets of a flow would constitute “processing.”

ii. “receiving a packet of the message”

Claim 1 further recites: “receiving a packet of the message and a data type of the message.” Under Implicit’s apparent claim constructions, Decasper98 discloses this element.

As packets arrive, they are classified into flows based on the values of several header fields including “protocol,” “source port,” and “destination port.” *Id.* at 3, 5. “[P]rotocol” would

comprise a data type under Implicit's apparent claim constructions. Additionally, because "source" and/or "destination port" typically contains a well-known port number indicating the application above, this would also comprise a data type under Implicit's apparent claim constructions.⁶

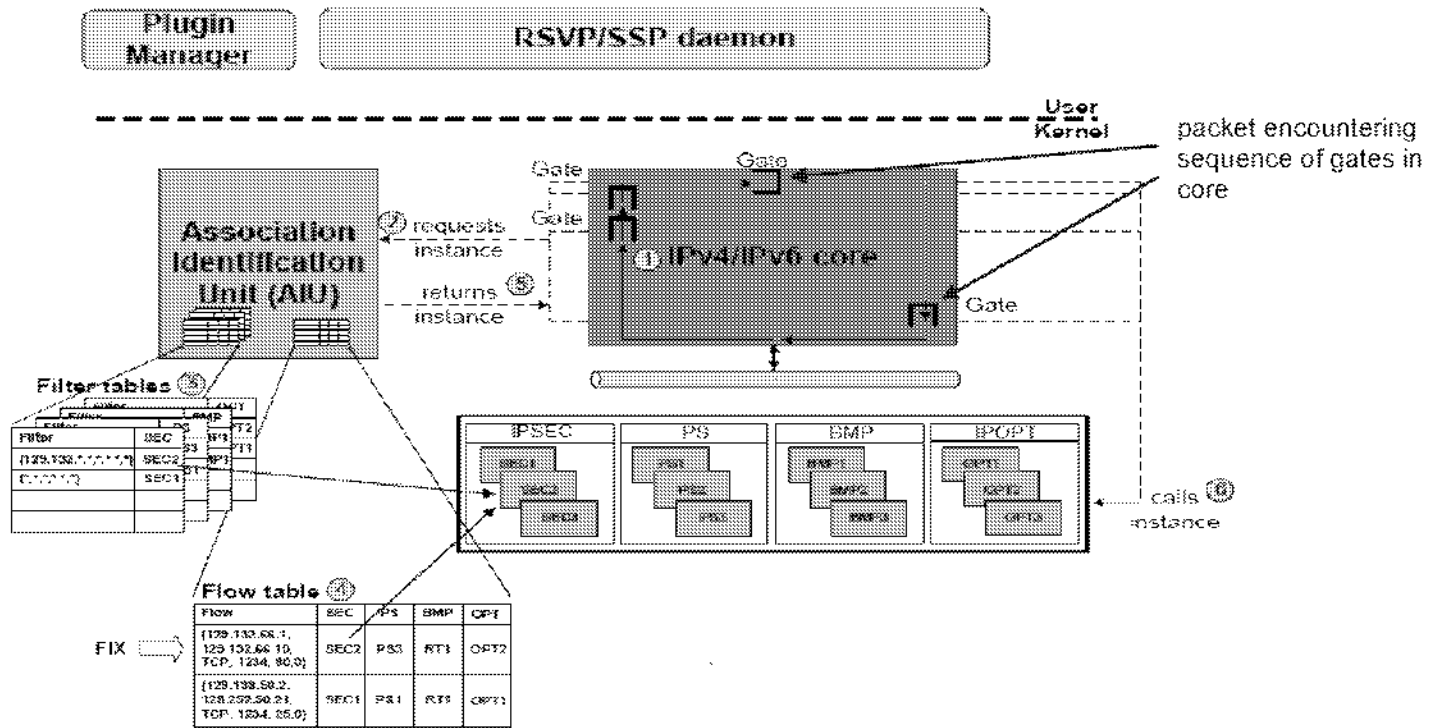
iii. *"dynamically identify a sequence of components"*

Claim 1 further recites: "analyzing the data type of a first packet of the message to dynamically identify a sequence of components for processing a plurality of packets of the message such that the output format of the components of the sequence match the input format of the next component in the sequence, wherein analyzing the data type of the first packet of the message to dynamically identify the sequence of components includes selecting individual components to form the sequence of components after the first packet of the message is received." Under Implicit's apparent claim constructions, Decasper98 discloses this element.

Before considering this element portion by portion, an overview is first presented of the manner in which the plugins for a particular flow are selected and recorded.

"[E]very single packet" processed by Decasper98 proceeds through a certain "data path" in the system's "core." *Id.* at 4. As a packet is moved through this "data path," it encounters a sequence of "gates." *Id.*

⁶ *See, e.g.*, Ex. 42 (RFC 1700) ("Assigned Numbers") (1994) at 16-19 ("WELL KNOWN PORT NUMBERS" including: "20 . . . File Transfer"; "23 . . . Telnet"; "41 . . . Graphics"; "58 . . . XNS Mail"; and so on. This reference is cited solely to help explain Decasper98. *See* MPEP § 2205.



Id. at 5 (Figure 3: “System Architecture and Data Path,” showing packet encountering a sequence of “Gate[s]” as it moves through system “core”). “A *gate* is a point in the IP core where the flow of execution branches off to an instance of a plugin.” *Id.* at 4 (emphasis in original). The correct plugin instance for processing a packet at a particular gate is determined by consulting a “filter table.” *Id.* at 5. “Filter tables store the bindings between filters and plugins for each gate.” *Id.* “[T]here is one filter table for every gate” in the system,” and the “filter table lookup algorithm finds the most specific . . . filter” matching the packet and “returns the corresponding plugin instance”—which is then invoked to perform the processing on that packet for that gate. *Id.* at 7, 5.

Filter table lookups are “slow[]” and “expensive,” and the “processing” of a packet in a system with “*n* gates” would require “*n* filter table lookups.” *Id.* at 5. It is for this reason that these filter table lookups are performed **only for the first packet of each flow**. *Id.* at 5, 3.

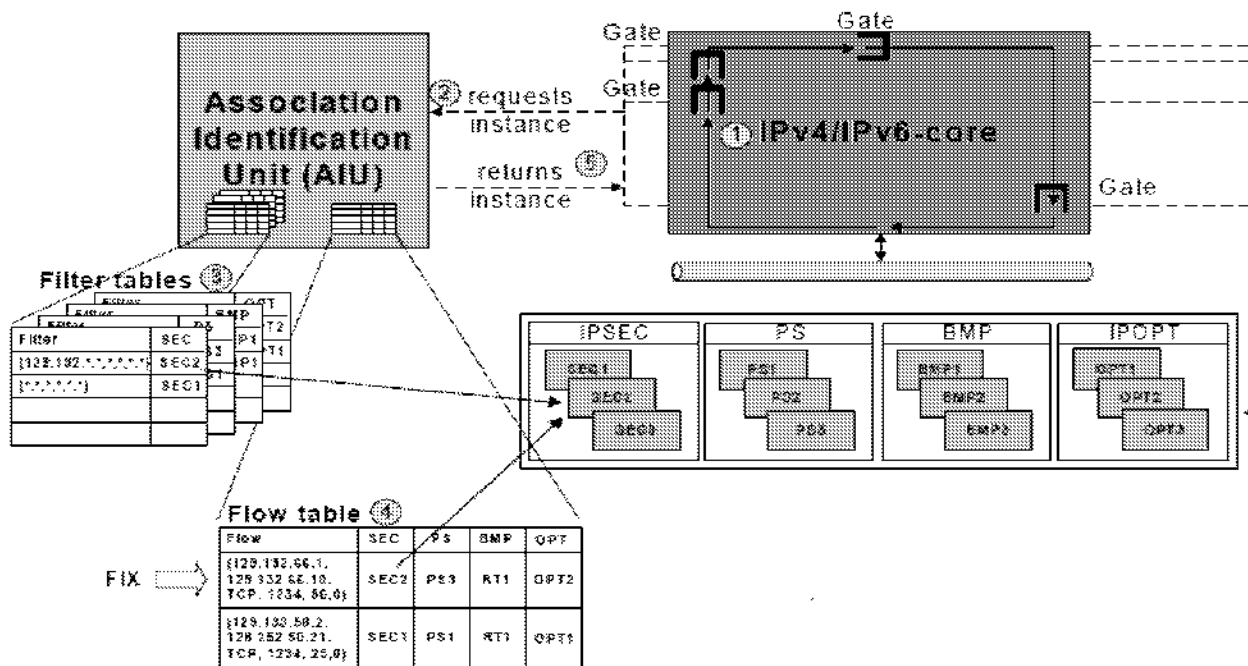
When “the first packet of a new flow” arrives, “an entry” is created for it in a “flow table.” *Id.* at

5. Each entry in the flow table “includes space for” a “pointer” to the correct “plugin instance” for “each gate that is implemented in the core.” *Id.* at 9. As the first packet encounters the sequence of n gates, pointers to the correct plugin instances returned by the n filter table lookups are **recorded** in the flow table entry. *Id.* at 9, 5.

Usually, filter table lookups are much slower than flow table lookups. An entry for a flow in the flow table serves as a fast cache for future lookups of packets belonging to that flow. Each flow table entry stores pointers to the appropriate plugins for all gates that can be encountered by packets belonging to the corresponding flow.

Id. at 5. As “subsequent packets” of the flow arrive and encounter the sequence of n gates in the “data path,” it is not necessary to perform a slow and expensive “filter table lookup” at each gate in order to obtain the correct plugin instance—because pointers to the entire sequence of plugins were recorded in the flow’s table entry as the first packet in the flow was processed. *Id.* at 4-5.

A portion of Figure 3 shows the above framework in action:



Id. at 5 (portion of Figure 3). There are four “Gate[s]” in this sample system’s data path, each with its own “Filter table[]” on the left. *Id.* There are two flows, each with an entry in the “Flow table.” *Id.* Because there are four gates, each flow entry contains four pointers to plugin instances. *Id.* As the first packet of the first flow encounters each gate in turn on the data path, the “Filter table” for that gate is applied to the packet to determine the correct plugin instance for that gate, which is recorded in the flow’s table entry. *Id.* For example, at an IP security gate (“SEC”/“IPSEC”), the plugin instance “SEC2” was chosen on the basis of the “SEC” filter table on the left, and a pointer to this “SEC2” plugin instance was recorded in the first flow entry (in column “SEC”). *Id.* Similar processing was applied at the other gates, with the result that the plugin sequence SEC2, PS3, RT1, OPT2 was chosen for packets of the first flow, and recorded in its flow table entry. Similar processing was applied to the first packet of the *second* flow, but because its data was different, a different sequence of plugins was obtained from the series of four filter table lookups: *i.e.*, the sequence SEC1, PS1, RT1, OPT1. *Id.*

With this general understanding of the plugin selection process in place, this claim element can now be considered portion by portion.

Claim 1 recites “analyzing the data type of a first packet of the message to dynamically identify a sequence of components for processing a plurality of packets of the message.” As explained above, as the first packet of a flow traverses the data path, a sequence of plugin instances is selected for the flow based on a series of filter table lookups. *Id.* at 4-5. Filter table filters match against only six aspects of the packet: “the six-tuple <*source address, destination address, protocol, source port, destination port, incoming interface*>.” *Id.* at 7, 3. Because “*protocol*” and “*source port*” and/or “*destination port*” would each comprise a “data type” under

Implicit's apparent claim constructions, the plugins selected by the filter tables are identified by "analyzing the data type of a first packet of the message." *See* Claim 1(ii) above.

This identification of plugins is performed "dynamically" in at least two senses under Implicit's apparent claim constructions.

(a) "dynamically": first sense

Because not one but "multiple packet classification steps (filter table lookups)" are required to generate the sequence of plugins for a flow, this can lead "exponentially" to an enormous number of valid sequences, "even with very few installed filters." *See id.* at 7. The various valid 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 operations to packet data which was not available to the system until the packet had arrived. *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

approach leads to so many valid sequences that it is impossible to even indicate them ahead of time in memory—since they would not fit.

Instead, Decasper98 adopts an algorithmic approach where the sequence of plugins for a flow is identified *dynamically* on demand, by applying the sequence of multiple filters to the first packet of the flow when it arrives.

(b) “dynamically”: second sense

Implicit has characterized the “dynamically identify” 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 9 (emphasis added).

Decasper98 discloses “dynamically identify” 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 this “dynamically identify” element as well. *Id.* at 9.

Thus, in at least two senses, Decasper98 discloses that the identification of plugin components is performed “dynamically,” under Implicit’s apparent claim constructions.

Claim 1 also recites “the output format of the components of the sequence match the input format of the next component in the sequence.” Decasper98 discloses various plugin components which could modify the format of a packet. For example, “plugins for IP security” perform “authentication and/or encryption.” *Id.* at 2, 5. Encryption converts packet data to an encrypted format, and performing encryption or authentication can add a header to the packet⁷ which would also change the packet’s format under Implicit’s apparent claim constructions. As another example, “plugins implementing IPv6 options” can add or remove IPv6 option headers

⁷ See, e.g., Ex. 25 (RFC 1825) (“Security Architecture for the Internet Protocol”) (1995) at 3 (citing “two specific headers . . . used to provide security services in IPv4 and IPv6” which may be added to a packet: an “IP Authentication Header” and an “IP Encapsulating Security Payload [encryption] . . . header.” RFC 1925 is cited by Decasper98 to explain its “plugins for IP security.” Ex. 25 (Decasper98) at 2 (“plugins for IP security” citing footnote “2”), 12 (footnote 2 citing “RFC 1825”). It is cited in this context solely to help explain Decasper98. See MPEP § 2205.

in the course of processing a packet.⁸ Ex. 25 (Decasper98) at 4. Certain other plugin components perform processing on a packet while leaving its format unchanged: *e.g.*, “a statistics gathering plugin.” *Id.* at 4. When applying any of the plugins of Decasper98 to a system, one of ordinary skill in the art would understand the sequence of gates in the system’s data path would be arranged in a compatible manner, such that the output format of one plugin component would match the input format of the next. For example, if a first component accepts packets in an unencrypted format (*e.g.*, so it may process their IPv6 headers) and a second component encrypts packets *in toto*, the gate for the second component in the data path would clearly need to follow that for the first, or the first component would be unable to perform its processing.

Claim 1 also recites “selecting individual components to form the sequence of components after the first packet of the message is received.” As explained above, a sequence of individual filter tables is applied to select the sequence of individual plugins for a flow, while the first packet of the flow traverses the data path. *Id.* at 5-7.

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

Claim 1 further recites: “storing an indication of each of the identified components so that the sequence does not need to be re-identified for subsequent packets of the message.” Under Implicit’s apparent claim constructions, Decasper98 discloses this element.

As explained above, each entry in the flow table “includes space for” a “pointer” to the correct “plugin instance” for “each gate that is implemented in the core.” *Id.* at 9. As the first

⁸ See, *e.g.*, Ex. 29 (Huitema) (“IPv6: The New Internet Protocol”) (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.

packet of a new flow encounters each gate on the data path, the system “performs a lookup” in the gate’s “filter table” to obtain the correct “plugin instance pointer” for the gate, which is then “store[d]” in the “flow table.” *Id.* at 5. “Subsequent packets” of the flow obtain these pointers not by performing filter table lookups of their own but by reading the pointers from the flow table, “which temporarily stores the information gathered by processing the first packet.” *Id.* at 3, 5, 9. Since “filter table lookups are much slower than flow table lookups,” “[s]ubsequent packets can benefit from faster lookup times.” *Id.* at 5.

v. “state information”

Claim 1 finally recites: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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 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. 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 an initial matter, it is important to observe that Decasper98 *expects* its individual plugin components to maintain state across packets, and it makes explicit provision for this in its architecture. Each entry in the flow table “includes space for . . . [a] pair of pointers for each gate that is implemented in the core. One pointer points to the plugin

instance that has been bound to the flow.⁹ The second points to *private data for that plugin instance*; it is used by the plugins to store *per-flow ‘soft’ state*.” Ex. 25 at 9 (emphasis added). Thus, Decasper98 provides each individual plugin an area for maintaining its own state information, and does so a “per-flow” basis: *i.e.*, if a particular plugin instance appeared in two flows, it would be given two such areas, so it could maintain a different set of state information for each flow. *See id.*

Some particular ways in which individual components would maintain state information across packets in the manner recited by claim 1 are discussed below. 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

⁹ *I.e.*, “storing an indication of the identified component[]” per claim 1, as discussed above.

“Lifetime of [the] Security Association.” Ex. 9 (RFC 1825) at 5-6.¹⁰ Under Implicit’s apparent claim constructions, maintaining such state information would read on this “state information” claim element.

(b) IPv6 options components

As a second group of examples, Decasper98 teaches multiple “plugins implementing IPv6 options,” and cites the IPv6 specification (RFC 1883). Ex. 25 at 4, 12 (citation to “RFC 1883”: “Internet Protocol, Version 6 (IPv6) Specification”). RFC 1883 explains how state information is maintained on a per-flow basis to support IPv6 options. *See* 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”).¹¹ Under Implicit’s apparent claim constructions, maintaining such state information would read on this “state information” claim 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

¹⁰ RFC 1825 is cited by Decasper98, and is cited in this context solely to help explain Decasper98. *See* MPEP § 2205.

¹¹ RFC 1883 is cited by Decasper98, and is cited in this context solely to help explain Decasper98. *See* MPEP § 2205.

constructions, maintaining such state information would read on this “state information” claim 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.” Ex. 25 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” claim element.

(b) Claim 4

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

Claim 4 recites: “A method in a computer system for processing a message, the message having a plurality of headers, the method comprising” Under Implicit’s apparent claim constructions, Decasper98 discloses this element.

As explained above, as packets arrive, they are classified into flows based on the following packet header fields: “*<source address, destination address, protocol, source port, destination port . . . >*.” Ex. 25 at 3, 5. The first three fields are found in an IP packet’s layer 3 header, and the final two in its layer 4 header (*e.g.*, TCP or UDP).¹²

¹² See, *e.g.*, Ex. 41 (RFC 791) (IP Specification) (1981) at 11 (“Source Address”; “Destination Address”; “Protocol”); Ex. 9 (RFC 793) (TCP Specification) (1981) at 15 (“Source Port”; “Destination Port”). These references are cited in this context solely to help explain Decasper98. See MPEP § 2205.

Other aspects of this claim element are discussed above. *See* Claim 1(i) above.

ii. “dynamically identify a sequence”

Claim 4 further recites: “analyzing the plurality of headers of a first packet of the message to dynamically identify a sequence of components for processing a plurality of packets of the message such that the output format of the components of the sequence match the input format of the next component in the sequence, wherein analyzing the plurality of headers of the first packet of the message to dynamically identify the sequence of components includes selecting individual components to form the sequence of components after the first packet of the message is received.” Under Implicit’s apparent claim constructions, Decasper98 discloses this element.

Regarding the limitation “*analyzing the plurality of headers of a first packet of the message to . . . identify a sequence of components,*” Decasper98 teaches that as the first packet of a flow traverses the data path, a sequence of plugin instances is selected for the flow based on a series of filter table lookups. Ex. 25 at 4-5. Filter table filters match against only six aspects of the packet: “the six-tuple <*source address, destination address, protocol, source port, destination port, incoming interface*>.” *Id.* at 7, 3. The first three are found in an IP packet’s layer 3 header, and the following two are found in its layer 4 header.

Other aspects of this claim element are discussed above. *See* Claim 1(iii) above.

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

Claim 4 further recites: “storing an indication of each of the identified components so that the 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* Claim 1(iv) above.

iv. “state information”

Claim 4 finally recites: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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 element. *See* Claim 1(v) above.

(c) Claim 10

i. “A computer readable storage medium”

Claim 10 recites in pertinent part: “A computer readable storage medium, other than a data transmission medium, containing instructions for processing packets of a message, the instructions comprising at least one computer-executable module configured to” 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 storage medium, other than a data transmission medium”: *e.g.*, from a hard disk in the device. Other aspects of this claim element are discussed above. *See* Claim 1(i) above.

ii. Other claim elements

The remaining elements of claim 10 are also disclosed by Decasper98. *See* Claim 1 above.

2. Decasper98 Renders Obvious Claims 1, 4, and 10 Under § 103

If certain aspects recited in claims 1, 4, and 10 of the ‘857 patent are not deemed to be disclosed or inherent over Decasper98, then the inclusion of those aspects certainly would be

obvious over Decasper98 in light of the background knowledge of one of ordinary skill in the art, under 35 U.S.C. § 103.

(a) *Claim 1*

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

Claim 1 recites: “A method in a computer system for processing packets of a message, 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 in a “computer system.” Other aspects of this claim element are discussed above. See Section V.A.1 (Decasper98 102) at Claim 1(i) above.

ii. “receiving a packet of the message”

Claim 1 further recites: “receiving a packet of the message and a data type of the message.” Under Implicit’s apparent claim constructions, Decasper98 renders obvious this element.

As packets arrive, they are classified into flows based on the values of several header fields including “protocol,” “source port,” and “destination port.” *Id.* at 3, 5. “[P]rotocol” would comprise a data type under Implicit’s apparent claim constructions. Because well-known port numbers are typically used to indicate the application above, it was obvious the packet would contain a source and/or destination port number which would comprise “a data type,” under Implicit’s apparent claim constructions.¹³

¹³ See, e.g., Ex. 42 (RFC 1700) (“Assigned Numbers”) (1994) at 16-19 (“WELL KNOWN PORT NUMBERS” including: “20 . . . File Transfer”; “23 . . . Telnet”; “41 . . .

iii. “dynamically identify a sequence of components”

Claim 1 further recites: “analyzing the data type of a first packet of the message to dynamically identify a sequence of components for processing a plurality of packets of the message such that the output format of the components of the sequence match the input format of the next component in the sequence, wherein analyzing the data type of the first packet of the message to dynamically identify the sequence of components includes selecting individual components to form the sequence of components after the first packet of the message is received.” Under Implicit’s apparent claim constructions, Decasper98 renders obvious this element.

Regarding the limitation “such that the output format of the components of the sequence match the input format of the next component,” one of ordinary skill in the art would understand the sequence of gates in the system’s data path should be arranged in a compatible manner, such that the output format of one plugin component would match the input format of the next. For example, if a first component accepts packets in an unencrypted format (*e.g.*, so it may process their IPv6 headers) and a second component encrypts packets *in toto*, it was obvious to position the gate for the second component after the gate for the first, since otherwise the first component would be unable to perform its processing. *See* Ex. 25 at 4-5.

Other aspects of this claim element are discussed above. *See* Section V.A.1 (Decasper98 102) at Claim 1(iii) above.

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

Claim 1 further recites: “storing an indication of each of the identified components so that the sequence does not need to be re-identified for subsequent packets of the message.” Under

Graphics”; “58 . . . XNS Mail”; and so on. This reference is cited solely to help explain Decasper98. *See* MPEP § 2205.

Implicit's apparent claim constructions, Decasper98 discloses this element. *See* Section V.A.1 (Decasper98 102) at Claim 1(iv) above.

v. "state information"

Claim 1 finally recites: "for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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 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. 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 an initial matter, it is important to observe that Decasper98 *expects* its individual plugin components to maintain state across packets, and it makes explicit provision for this in its architecture. Each entry in the flow table "includes space for [a] pair of pointers for each gate that is implemented in the core. One pointer points to the plugin instance that has been bound to the flow.¹⁴ The second points to *private data for that plugin instance*; it is used by the plugins to store *per-flow 'soft' state*." Ex. 25 at 9 (emphasis added). Thus, Decasper98 provides each individual plugin an area for maintaining its own state

¹⁴ *I.e.*, "storing an indication of the identified component[]" per claim 1, as discussed above.

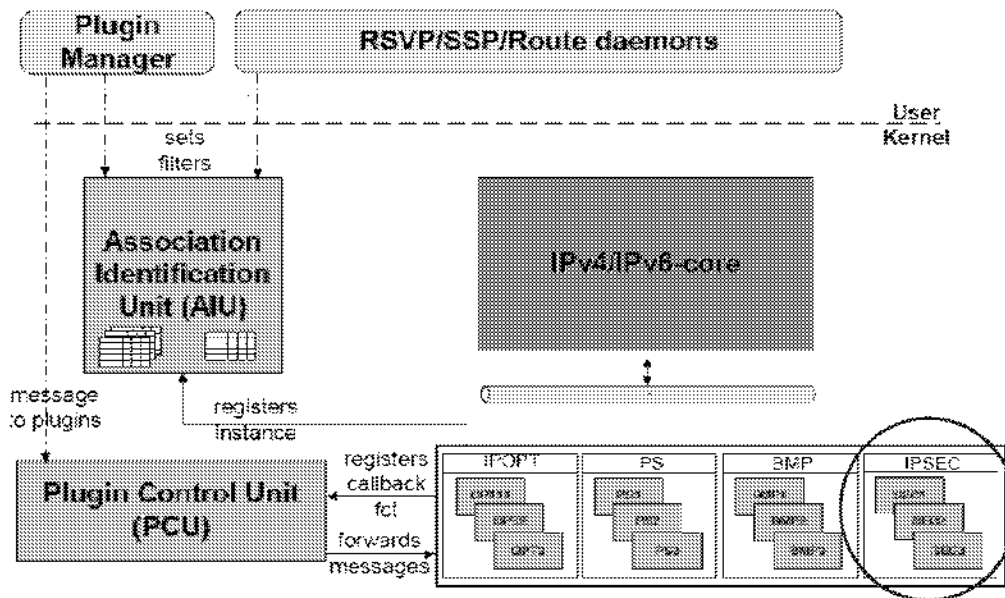
information, and does so a “per-flow” basis: *i.e.*, if a particular plugin instance appeared in two flows, it would be given two such areas, so it could maintain a different set of state information for each flow. *See id.*

Some particular ways in which it was obvious for individual components to maintain state information across packets in the manner recited by claim 1 are discussed below. Since Decasper98 teaches that its plugin components are selected on the basis of separate, independent filter tables, it was obvious for virtually any combination of these components to 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”).¹⁵

¹⁵ 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).



Ex. 25 at 4 (Figure 2, showing multiple “IPSEC” plugins: “SEC1 SEC2 SEC3”).

Since encryption and authentication are independent operations which need not be applied to the same packet, it was obvious to provide distinct plugin components for encryption and authentication. *See, e.g.,* Ex. 26 (RFC 1825) at 8 (“The two IP security mechanisms [authentication and encryption] may be used together or separately”).

RFC 1825 explains that various forms of state information would be maintained by these plugin 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. 26 (RFC 1825) at 5-6. Obvious implementations to maintain this state information would read on this “state information” claim element, under Implicit’s apparent claim constructions. For example, both the encryption and authentication component(s) 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 encryption plugin component(s) in particular, an obvious implementation of the encryption algorithm would read on this claim element in still another manner, under Implicit’s apparent claim constructions. Decasper98 supports “the Internet protocol stack,” and one of ordinary skill would be aware of common techniques for implementing an encryption algorithm which “MUST” be supported by the security architecture for the Internet Protocol: *i.e.*, the “ESP DES-CBC” algorithm described in RFC 1829.¹⁶ Ex. 25 (Decasper98) at 7. In order to apply this required 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). Ex. 27 (RFC 1829) at 1. RFC 1829 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 common technique, and find it obvious to apply to an encryption component of Decasper98. Doing so would entail, for each packet, retrieving the previous counter value, applying it to encrypt the packet, 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.

¹⁶ See Ex. 25 (RFC 1825) (“Security Architecture for the Internet Protocol”) 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”). Again, RFC 1825 and RFC 1829 are cited in this section solely to help explain Decasper98. See MPEP § 2205.

(b) IPv6 options components

As a second group of examples, Decasper98 teaches multiple “plugins implementing IPv6 options,” and cites the IPv6 specification (RFC 1883). Ex. 25 at 4, 12 (citation to “RFC 1883”: “Internet Protocol, Version 6 (IPv6) Specification”). RFC 1883 explains how state information is 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”).¹⁷ One of ordinary skill in the art would therefore be familiar with the technique and find it obvious to apply to Decasper98, which also employs a “flow”-based architecture. Under Implicit’s apparent claim constructions, such maintenance of state information would read on this “state information” claim 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 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

¹⁷ RFC 1883 is cited by Decasper98, and is cited in this context solely to help explain Decasper98. *See* MPEP § 2205.

apparent claim constructions, such maintenance of state information would read on this “state information” claim 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 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” claim 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.¹⁸ Under Implicit’s apparent claim constructions, such maintenance of state information of state information would read on this “state information” claim element.

(b) Claim 4

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

Claim 4 recites: “A method in a computer system for processing a message, the message having a plurality of headers, the method comprising” Under Implicit’s apparent claim constructions, Decasper98 renders obvious this element.

¹⁸ See, for example, the Shwed and Checkpoint references cited herein.

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 in a “computer system.” Other aspects of this claim element are discussed above. *See* Section V.A.1 (Decasper98 102) at Claim 4(i) above.

ii. “dynamically identify a sequence”

Claim 4 further recites: “analyzing the plurality of headers of a first packet of the message to dynamically identify a sequence of components for processing a plurality of packets of the message such that the output format of the components of the sequence match the input format of the next component in the sequence, wherein analyzing the plurality of headers of the first packet of the message to dynamically identify the sequence of components includes selecting individual components to form the sequence of components after the first packet of the message is received.” Under Implicit’s apparent claim constructions, Decasper98 renders obvious this element.

Regarding the limitation “such that the output format of the components of the sequence match the input format of the next component,” *see* Claim 1(iii) above. Other aspects of this claim element are discussed elsewhere above. *See* Section V.A.1 (Decasper98 102) at Claim 4(ii).

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

Claim 4 further recites: “storing an indication of each of the identified components so that the 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.A.1 (Decasper98 102) at Claim 4(iii) above.

iv. “state information”

Claim 4 finally recites: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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 element. *See* Claim 1(v) above.

(c) Claim 10

i. “A computer readable storage medium”

Claim 10 recites: “A computer readable storage medium, other than a data transmission medium, containing instructions for processing packets of a message, the instructions comprising at least one computer-executable module configured to” 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 at least obvious that such software modules would be dynamically loaded from a “computer readable storage medium, other than a data transmission medium”—such as a hard disk in the device. Other aspects of this claim element are discussed above. *See* Claim 1(i) above.

ii. Other claim elements

The remaining elements of claim 10 are also disclosed or rendered obvious by Decasper98. *See* Claim 1 above.

3. Decasper98 in View of RFC 1825 and RFC 1829 Renders Obvious Claims 1, 4, and 10 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 ‘857 patent.

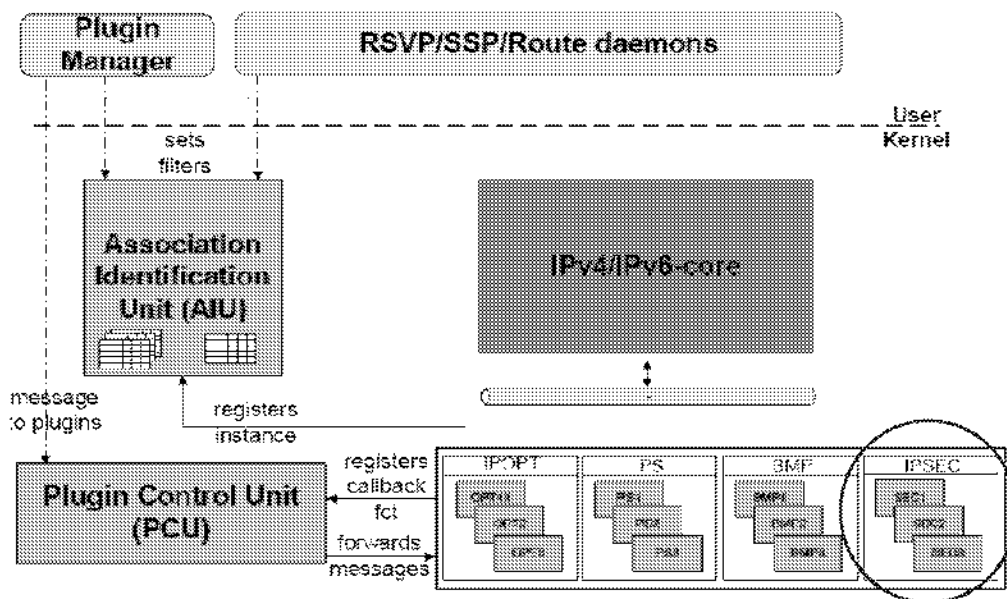
If certain aspects recited in claims 1, 4, and 10 of the ‘857 patent are not deemed to be disclosed, inherent, or obvious over Decasper98 alone, then the inclusion of those aspects certainly would be obvious over Decasper98 in view of RFC 1825 and RFC 1829 in light of the background knowledge of one of ordinary skill in the art, 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”).

(a) Claim 1

Claim 1 recites in pertinent part: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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.

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”).



Ex. 25 at 4 (Figure 2, showing multiple “IPSEC” plugins: “SEC1 SEC2 SEC3”).

Because there are multiple distinct IP security operations which may be performed (*e.g.*, encryption and/or authentication) and these operations are independent and need not be applied to the same packet, it was obvious to provide distinct plugin(s) for authentication and encryption. *See, e.g.*, Ex. 26 (RFC 1825) at 8 (“The two IP security mechanisms [authentication and encryption] may be used together or separately”).

RFC 1825 explains that various forms of state information would be maintained by these plugin 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. 26 (RFC 1825) at 5-6. Obvious implementations to maintain this state information would read on this “state information” claim element, under Implicit’s apparent claim constructions. For example, both the encryption and authentication component(s) 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 plugin component(s) in particular, an obvious implementation of the encryption algorithm 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 this required 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). 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 “common, acceptable” counter technique to an encryption component of Decasper98. Doing so would entail, for each packet, retrieving the previous counter value, applying it to encrypt the packet, 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.

To summarize, Decasper98 in view of RFC 1825 and RFC 1829 renders obvious distinct plugin components for encryption and authentication which would maintain state information across packets in the manner recited by claim 1. Considering these components either together with each other or as combined with other stateful plugins discussed elsewhere above (*e.g.*, IPv6 options components, a statistics gathering component, a packet scheduling component, a firewall component), they would comprise “a plurality of components” as recited by this claim element. *See* Section V.A.2 (Decasper98 103) at Claim 1(v) above. Since Decasper98 teaches that its plugin components are selected on the basis of separate, independent filter tables, it was obvious for virtually any combination of these components to be applied to a particular flow.

(b) Claim 4

Claim 4 recites in pertinent part: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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 element. *See* Claim 1 above.

(c) Claim 10

Claim 10 recites in pertinent part: “for each of a plurality of components in the identified sequence: perform the processing of each packet by the identified component; and store 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 element. *See* Claim 1 above.

4. Decasper98 in View of RFC 1883 Renders Obvious Claims 1, 4, and 10 Under § 103

The specification RFC 1883 (“Internet Protocol, Version 6 (IPv6) Specification”) (Exhibit 28, “RFC 1883”) by S. Deering *et al.* was published in December 1995. It was not considered during the prosecution of the ‘857 patent.

If certain aspects recited in claims 1, 4, and 10 of the ‘857 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 in light of the background knowledge of one of ordinary skill in the art, under 35 U.S.C. § 103.

It was obvious to supplement the teachings of Decasper98 with RFC 1883, because Decasper98 discloses “plugins implementing IPv6 options” and expressly cites to RFC 1883, which explains IPv6 options. Ex. 25 (Decasper98) at 4 (“plugins implementing IPv6 options”), 12 (citation to “RFC 1883”).

(a) Claim 1

Claim 1 recites in pertinent part: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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 renders obvious this “state information” claim element.

Decasper98 teaches multiple “plugins implementing IPv6 options.” Ex. 25 at 4 (including Figure 2, showing multiple “IPOPT” plugins: “OPT1 OPT2 OPT3”). Decasper98 cites the IPv6 specification (RFC 1883), which explains how state information is maintained on a per-flow basis to support IPv6 options. *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 a “flow”-based architecture and RFC 1883 explicitly discloses this flow-based technique for processing IPv6 options, it was at least obvious to apply the technique to Decasper98’s IPv6 options plugins. Under Implicit’s apparent claim constructions, such maintenance of state information would read on this “state information” claim element.

As combined with each other and/or other stateful plugins discussed elsewhere above (*e.g.*, IP security components, a statistics gathering component, a packet scheduling component, a firewall component), such stateful IPv6 option component(s) would comprise “a plurality of components” as recited by this claim element. *See* Section V.A.2 (Decasper98 103) at Claim 1(v) above. Since Decasper98 teaches that its plugin components are selected on the basis of separate, independent filter tables, it was obvious for virtually any combination of these components to be applied to a particular flow.

(b) Claim 4

Claim 4 recites in pertinent part: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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 renders obvious this element. *See* Claim 1 above.

(c) Claim 10

Claim 10 recites in pertinent part: “for each of a plurality of components in the identified sequence: perform the processing of each packet by the identified component; and store 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 renders obvious this element. *See* Claim 1 above.

5. Decasper98 in View of Decasper97 Renders Obvious Claims 1, 4, and 10 Under § 103

The article “Crossbow: A Toolkit for Integrated Services over Cell Switched IPv6” (Exhibit 30, “Decasper97”) by Dan Decasper *et. al* was published on May 28, 1997. Decasper97 was not considered during the prosecution of the ‘857 patent.

If certain aspects recited in claims 1, 4, and 10 of the ‘857 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 in light of the background knowledge of one of ordinary skill in the art, 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 308 (“Flow entries”), 307 (“dynamically loadable modules”), 307-08 (filter

operation), 307-08 (modules include “authentication modules . . . encryption modules . . . IPv6 option modules . . . and packet scheduling modules.”).

Because Decasper98 and Decasper97 are similar in approach and detail, Decasper97 confirms the obviousness of claims 1, 4, and 10 in a number of ways. One particularly pertinent aspect of Decasper97 is pointed out below.

(a) Claim 1

Claim 1 recites in pertinent part: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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” claim element.

As explained above, it was obvious over Decasper98 alone to employ distinct plugin components for encryption and authentication, since encryption and authentication are independent operations which need not be applied to the same packet. *See* Section V.A.2 (Decasper98 103) at Claim 1(v)(a) above. Decasper97 renders this even more obvious by teaching precisely that: “Five different module types are supported in the initial version,” including “authentication modules” and “encryption modules.” Ex. 30 at 307.

Stateful encryption and authentication algorithms are commonplace, and one of ordinary skill in the art would have found it obvious to apply such stateful algorithms to these distinct encryption and authentication components. For example, one of ordinary skill in the art would be aware that for both stateful encryption¹⁹ and stateful authentication²⁰, a counter is typically

¹⁹ *See, e.g.*, Ex. 17 (Bellare97) (“A Concrete Security Treatment of Symmetric Encryption”) (1997) at 397 (“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

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.

To summarize, Decasper98 in view of Decasper97 renders obvious distinct plugin components for encryption and authentication which would maintain state information across packets in the manner recited by claim 1. Considering these components either together with each other or as combined with other stateful plugins discussed elsewhere above (*e.g.*, IPv6 options components, a statistics gathering component, a packet scheduling component, a firewall component), they would comprise “a plurality of components” as recited by this claim element. *See* Section V.A.2 (Decasper98 103) at Claim 1(v) above. Since Decasper98 teaches that its plugin components are selected on the basis of separate, independent filter tables, it was obvious for virtually any combination of these components to be applied to a particular flow.

(b) Claim 4

Claim 4 recites in pertinent part: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; and storing state information relating to the processing of the component with the packet for use when

encryption”). This reference is cited in this context solely to help explain the prior art. *See* MPEP § 2205.

²⁰ *See, e.g.*, Ex. 18 (Bellare95) (“XOR MACs: New Methods for Message Authentication Using Finite Pseudorandom Functions”) (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.

processing the next packet of the message.” Under Implicit’s apparent claim constructions, Decasper98 in view of Decasper97 renders obvious this element. *See* Claim 1 above.

(c) Claim 10

Claim 10 recites in pertinent part: “for each of a plurality of components in the identified sequence: perform the processing of each packet by the identified component; and store 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 Decasper97 renders obvious this element. *See* Claim 1 above.

6. Decasper98 in View of Decasper97, Bellare97, and Bellare95 Renders Obvious Claims 1, 4, and 10 Under § 103

The article “A Concrete Security Treatment of Symmetric Encryption” (Exhibit 17, “Bellare97”) by M. Bellare *et al.* was published in 1997. The article “XOR MACs: New Methods for Message Authentication Using Finite Pseudorandom Functions” (Exhibit 18, “Bellare95”) by M. Bellare *et al.* was published in 1995. Neither was considered during the prosecution of the ‘857 patent.

If certain aspects recited in claims 1, 4, and 10 of the ‘857 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 in light of the background knowledge of one of ordinary skill in the art, 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 invited and supported by Decasper98.

(a) Claim 1

Claim 1 recites in pertinent part: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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” claim 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_i 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.

To summarize, Decasper98 in view of Decasper97, Bellare97, and Bellare95 renders obvious distinct plugin components for encryption and authentication which would maintain state information across packets in the manner recited by claim 1. Considering these components either together with each other or as combined with other stateful plugins discussed elsewhere above (*e.g.*, IPv6 options components, a statistics gathering component, a packet scheduling component, a firewall component), they would comprise “a plurality of components” as recited by this claim element. *See* Section V.A.2 (Decasper98 103) at Claim 1(v) above. Since Decasper98 teaches that its plugin components are selected on the basis of separate, independent filter tables, it was obvious for virtually any combination of these components to be applied to a particular flow.

(b) Claim 4

Claim 4 recites in pertinent part: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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 element. *See* Claim 1 above.

(c) Claim 10

Claim 10 recites in pertinent part: “for each of a plurality of components in the identified sequence: perform the processing of each packet by the identified component; and store 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 element. *See* Claim 1 above.

7. Decasper98 in View of IBM96 Renders Obvious Claims 1, 4, and 10 Under § 103

The book “Local Area Network Concepts and Products: Routers and Gateways” (Exhibit 19, “IBM96”) was published by IBM in May 1996. IBM96 was not considered during prosecution of the ‘857 patent.

If certain aspects recited in claims 1, 4, and 10 of the ‘857 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 in light of the background knowledge of one of ordinary skill in the art, 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

Claim 1 recites in pertinent part: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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” claim 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, additional plugin(s) implementing compression would be exactly the sort of extensions invited and expected by Decasper98.

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.²¹ 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.²²

To summarize, Decasper98 in view of IBM96 renders obvious compression plugin components employing “adaptive” algorithms which would maintain state information across packets in the manner recited by claim 1. As combined with other stateful plugins discussed elsewhere above (e.g., IP security components, IPv6 options components, a statistics gathering component, a packet scheduling component, a firewall component), they would comprise “a plurality of components” as recited by this claim element. See Section V.A.2 (Decasper98 103)

²¹ See, *e.g.*, Ex. 5 (Nelson) (“The Data Compression Book”) (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.

²² See, *e.g.*, Ex. 5 (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.

at Claim 1(v) above. Since Decasper98 teaches that its plugin components are selected on the basis of separate, independent filter tables, it was obvious for virtually any combination of these components to be applied to a particular flow.

(b) Claim 4

Claim 4 recites in pertinent part: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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 element. *See* Claim 1 above.

(c) Claim 10

Claim 10 recites in pertinent part: “for each of a plurality of components in the identified sequence: perform the processing of each packet by the identified component; and store 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 element. *See* Claim 1 above.

8. Decasper98 in View of IBM96 and Nelson Renders Obvious Claims 1, 4, and 10 Under § 103

The treatise “The Data Compression Book” (Exhibit 5, “Nelson”) by Mark Nelson *et al.* was published on November 6, 1995. Nelson was not considered during the prosecution of the ‘857 patent.

If certain aspects recited in claims 1, 4, and 10 of the ‘857 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 in light of the background knowledge of one of ordinary skill in the art, under 35 U.S.C. § 103.

It was obvious to supplement the teachings of Decasper98 and IBM96 with Nelson, because IBM96 discloses 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, additional plugins implementing compression algorithms would be exactly the sort of extensions invited and supported by Decasper98.

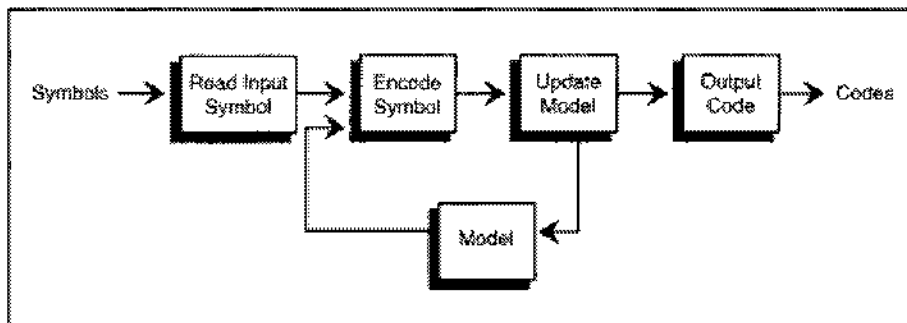
(a) Claim 1

Claim 1 recites in pertinent part: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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 and Nelson renders obvious this “state information” claim 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 adaptive algorithms would have been an obvious choice for compression components.

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.

As observed above, Nelson teaches a number of lossless, adaptive compression algorithms in Chapters 1 to 9 which would have been obvious choices to apply to compression

plugin components of Decasper98. *See id.* at 9 (“All of the compression techniques discussed through chapter 9 are ‘lossless’”).

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 choice for at least one compression plugin. *See* Ex. 19 (IBM96) at 95-96, 84. Nelson confirms this algorithm is stateful and “adaptive” in the manner described above. *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).

To summarize, Decasper98 in view of IBM96 and Nelson renders obvious compression plugin components employing “adaptive” algorithms which would maintain state information across packets in the manner recited by claim 1. As combined with other stateful plugins discussed elsewhere above (*e.g.*, IP security components, IPv6 options components, a statistics gathering component, a packet scheduling component, a firewall component), they would comprise “a plurality of components” as recited by this claim element. *See* Section V.A.2 (Decasper98 103) at Claim 1(v) above. Since Decasper98 teaches that its plugin components are selected on the basis of separate, independent filter tables, it was obvious for virtually any combination of components to be applied to a particular flow.

(b) Claim 4

Claim 4 recites in pertinent part: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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 and Nelson renders obvious this element. *See* Claim 1 above.

(c) *Claim 10*

Claim 10 recites in pertinent part: “for each of a plurality of components in the identified sequence: perform the processing of each packet by the identified component; and store 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 and Nelson renders obvious this element. *See* Claim 1 above.

9. Decasper98 in View of RFC 1825, RFC 1829, Decasper97, Bellare97, Bellare95, IBM96, and Nelson Renders Obvious Claims 1, 4, and 10 Under § 103

If certain aspects recited in claims 1, 4, and 10 of the ‘857 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 in light of the background knowledge of one of ordinary skill in the art, 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.”). Claims 1, 4, and 10 recite elements regarding “state information” as relating to “a plurality of components.”

Decasper98 teaches “plugins for IP security,” and **Decasper97** confirms the obviousness of providing separate IP security plugin components for encryption and authentication. **IBM96** confirms the obviousness of additional plugin components for compression.

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

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 stateful components types (encryption, authentication, compression) would be applied to a particular flow. This was 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”). Moreover, Decasper98 teaches additional stateful components which would read on these “state information” claim elements, including plugin components for statistics gathering, packet scheduling, and firewall functions—and it was obvious for any of these to be applied to a particular flow as well. *See* Sections V.C.1 (Decasper98 102) and V.C.2 (Decasper98 103) above.

Claims 1, 4, and 10 recite “dynamically identify a sequence of components.”

Decasper98 selects the sequence of plugin components for a flow on the basis of *multiple independent filter tables*. “[E]ven with very few installed filters,” this leads to “exponentially” many valid component sequences—so many, in fact, that it is “infeasible” to even indicate 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"—and including at least to the moment a new flow would begin. *Id.* at 9. This also reads on these "dynamic[]" claim elements, under Implicit's apparent claim constructions.

In short, there is no aspect of claims 1, 4, and 10 which was not obvious over the prior art and combinations cited herein.

10. Decasper98 in View of Fraser Renders Obvious Claims 1, 4, and 10 Under § 103

The publication "DTE Firewalls: Phase Two Measurement and Evaluation Report" (Exhibit 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 '857 patent.

If certain aspects recited in claims 1, 4, and 10 of the '857 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 in light of the background knowledge of one of ordinary skill in the art, 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 which permits an administrator to dynamically configure policies (expressed in filters) "even when network traffic is transiting

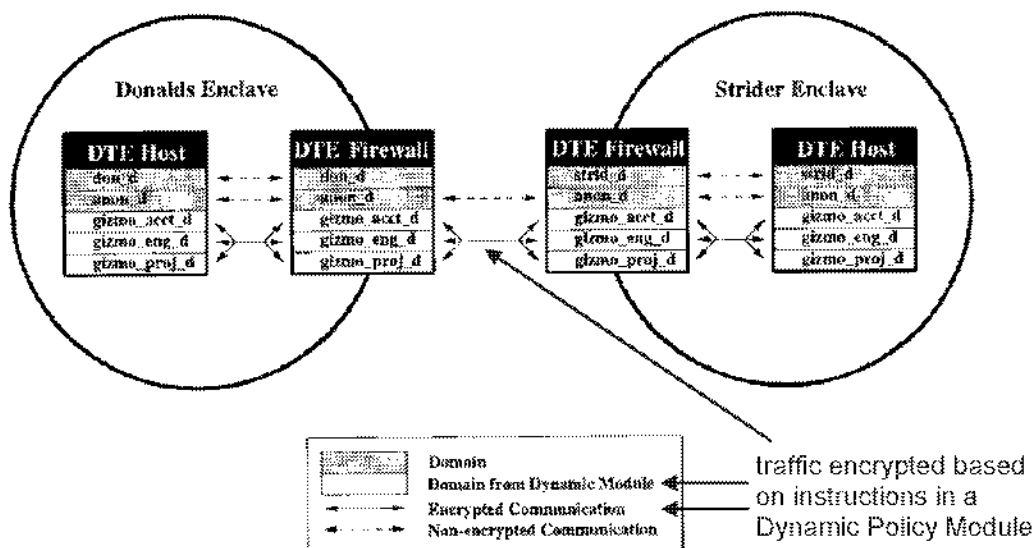
through the system” (Ex. 25 at 9), Fraser teaches a more comprehensive framework for such a capability, and provides additional detail on how such a framework would be implemented.

(a) Claim 1

Claim 1 recites in pertinent part: “analyzing the data type of a first packet of the message to dynamically identify a sequence of components for processing a plurality of packets of the message such that the output format of the components of the sequence match the input format of the next component in the sequence, wherein analyzing the data type of the first packet of the message to dynamically identify the sequence of components includes selecting individual components to form the sequence of components after the first packet of the message is received.” Under Implicit’s apparent claim constructions, Decasper98 in view Fraser renders obvious this element.

Decasper98 alone renders obvious these elements. *See* Section V.A.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 framework²³ 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 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

²³ 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”).

(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 identify a sequence of components for processing a plurality of packets of the message."

(b) Claim 4

Claim 4 recites in pertinent part: "analyzing the plurality of headers of a first packet of the message to dynamically identify a sequence of components for processing a plurality of packets of the message such that the output format of the components of the sequence match the input format of the next component in the sequence, wherein analyzing the plurality of headers of the first packet of the message to dynamically identify the sequence of components includes selecting individual components to form the sequence of components after the first packet of the

message is received.” Under Implicit’s apparent claim constructions, Decasper98 in view of Fraser renders obvious this element. *See* Claim 1 above.

(c) Claim 10

Claim 10 recites in pertinent part: “analyze the data type of a first packet of the message to dynamically identify a sequence of components for processing a plurality of packets of the message such that the output format of the components of the sequence match the input format of the next component in the sequence, wherein analyzing the data type of the first packet of the message to dynamically identify the sequence of components includes selecting individual components to form the sequence of components after the first packet of the message is received.” Under Implicit’s apparent claim constructions, Decasper98 in view of Fraser renders obvious this element. *See* Claim 1 above.

11. Decasper98 in View of Fraser, RFC 1825, and RFC 1829 Under § 103

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. As applied to the previous combination of Decasper98 in view of Fraser, RFC 1825 and RFC 1829 further confirm that “a plurality of components” in a sequence would maintain “state information” across packets in the manner recited by claims 1, 4, and 10. *See* Sections V.A.10 (Decasper98+Fraser) and V.A.3 (Decasper98+RFC 1825+RFC 1829) above.

12. Decasper98 in View of Bellissard Renders Obvious Claims 1, 4, and 10 Under § 103

The article “Dynamic Reconfiguration of Agent-Based Applications” (Exhibit 23, “Bellissard”) by Luc Bellissard *et al.* was published by September 10, 1998. Bellissard was not considered during prosecution of the ‘857 patent.

If certain aspects recited in claims 1, 4, and 10 of the '857 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 in light of the background knowledge of one of ordinary skill in the art, 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 (Ex. 25 at 2), and Bellissard teaches a technique for enhancing the dynamic extensibility of such an architecture, as will be explained below.

While Decasper98 already teaches a platform that allows administrators to 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

Claim 1 recites in pertinent part: “analyzing the data type of a first packet of the message to dynamically identify a sequence of components for processing a plurality of packets of the message such that the output format of the components of the sequence match the input format of the next component in the sequence, wherein analyzing the data type of the first packet of the message to dynamically identify the sequence of components includes selecting individual components to form the sequence of components after the first packet of the message is received.”

Under Implicit’s apparent claim constructions, Decasper98 alone renders obvious this element. *See* Section V.A.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 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 1-2.²⁴

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

²⁴ 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”).

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, 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 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 identify a sequence of components for processing a plurality of packets of the message.”

(b) Claim 4

Claim 4 recites in pertinent part: “analyzing the plurality of headers of a first packet of the message to dynamically identify a sequence of components for processing a plurality of packets of the message such that the output format of the components of the sequence match the input format of the next component in the sequence, wherein analyzing the plurality of headers of the first packet of the message to dynamically identify the sequence of components includes selecting individual components to form the sequence of components after the first packet of the message is received.”

Under Implicit's apparent claim constructions, Decasper98 alone renders obvious this element. *See* Section V.A.2 (Decasper98 103) at Claim 4. As applied to Decasper98, Bellissard further underscores the "dynamic[]" nature of the identification under Implicit's apparent claim constructions, as explained above. *See* Claim 1 above.

(c) Claim 10

Claim 10 recites in pertinent part: "analyze the data type of a first packet of the message to dynamically identify a sequence of components for processing a plurality of packets of the message such that the output format of the components of the sequence match the input format of the next component in the sequence, wherein analyzing the data type of the first packet of the message to dynamically identify the sequence of components includes selecting individual components to form the sequence of components after the first packet of the message is received." Under Implicit's apparent claim constructions, Decasper98 in view of Bellissard renders obvious this element. *See* Claim 1 above.

13. Decasper98 in View of Bellissard, RFC 1825, and RFC 1829 Under § 103

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. As applied to the previous combination of Decasper98 in view of Bellissard, RFC 1825 and RFC 1829 further confirm that "a plurality of components" in a sequence would maintain "state information" across packets in the manner recited by claims 1, 4, and 10. *See* Sections V.A.12 (Decasper98+Bellissard) and V.A.3 (Decasper98+RFC 1825+RFC 1829) above.

14. Decasper98 in View of Wetherall Renders Obvious Claims 1, 4, and 10 Under § 103

The article “The Active IP Option” (Exhibit 47, “Wetherall”) by David J. Wetherall and David L. Tennenhouse was published by September 11, 1996. It was not considered during the prosecution of the ‘857 patent.

If certain aspects recited in claims 1, 4, and 10 of the ‘857 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 Wetherall in light of the background knowledge of one of ordinary skill in the art, under 35 U.S.C. § 103.

It was obvious to supplement the teachings of Decasper98 with Wetherall because Decasper98 teaches an extensible flow-based architecture for routers, and Wetherall teaches that its “Active IP Option” technique is “a generic capability” that should be applied to “routers” and “[f]lows.” Ex. 25 (Decasper98) at 2; Ex. 47 (Wetherall) at 34-35.

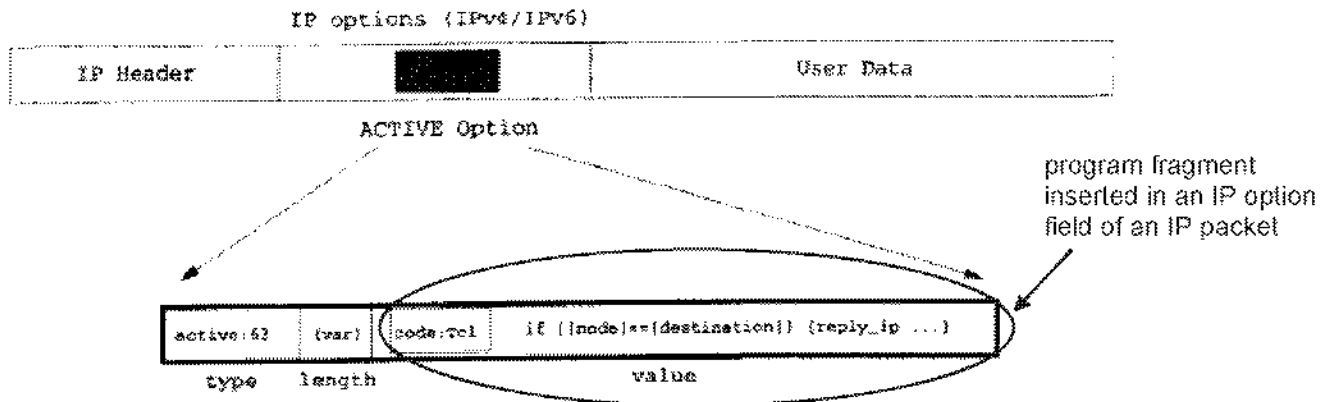
(a) Claim 1

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

Claim 1 recites: “A method in a computer system for processing packets of a message, the method comprising” Under Implicit’s apparent claim constructions, Decasper98 in view of Wetherall renders obvious this element.

Wetherall teaches an approach called “Active Networks,” which “break with tradition by allowing the network to perform customized computations on the user data.” Ex. 47 at 33. “For example, a user of an active network could send a customized video transcoding program to a node within the network (e.g., a router) and request that the node execute that program when processing their packets.” *Id.*

Wetherall “retrofit[s]” these “active capabilities” atop “the existing Internet” by exploiting the existing “options mechanism of the IP layer to carry program fragments.” *Id.* at 35. IP options are of flexible length and type (“generic type-length-value format of IP options”), and Wetherall simply defines a new option type “to carry program fragments, which may be encoded in a variety of languages.” *Id.* at 36.



Id. at 35 (Figure 1: “Format of the Active IP Option Field,” showing “code” embedded in an IP option field of a packet). Thereby, “passive packets of present day architecture” are replaced “with active ‘capsules’ – miniature programs that are executed at each router they traverse.” *Id.* at 34-35.

Both the already-discussed components of Decasper98 and the actively-delivered components of Wetherall would perform “processing” on the packets of a particular flow. *I.g.*, Ex. 47 (Wetherall) at 33 (“a user of an active network could send a customized video transcoding program to a node . . . and request that the node execute that program when processing *their packets.*”) (emphasis added).

ii. “receiving a packet of the message”

Claim 1 further recites: “receiving a packet of the message and a data type of the message.” Under Implicit’s apparent claim constructions, Decasper98 in view of Wetherall renders obvious this element.

In addition to the data type analysis performed by Decasper, Wetherall further requires that the data type of IP option field(s) in a packet be analyzed, to determine whether they contain code. *Id.* at 35-36.

iii. “dynamically identify a sequence of components”

Claim 1 further recites: “analyzing the data type of a first packet of the message to dynamically identify a sequence of components for processing a plurality of packets of the message such that the output format of the components of the sequence match the input format of the next component in the sequence, wherein analyzing the data type of the first packet of the message to dynamically identify the sequence of components includes selecting individual components to form the sequence of components after the first packet of the message is received.” Under Implicit’s apparent claim constructions, Decasper98 in view of Wetherall renders obvious this element.

Wetherall teaches that each packet may contain both “User data” and one or more programs in IP options field(s). *Id.* at 35 (“miniature programs,” and Figure 1 showing “User Data” plus “IP options” fields). 36 (“These fragments are . . . executed by active routers along the path taken by the datagram”).

Wetherall also teaches that “a user of an active network could send a customized video transcoding program to a node within the network (e.g., a router) and request that the node execute that program when processing their packets.” *Id.* at 33.

As applied to Decasper98, the first packet of a flow could contain one or more plugin components to be used for performing coding or other operations such as encryption on the packets of the flow. Decasper98 was an especially obvious target for this treatment, because it repeatedly emphasizes that the “extensibility” of its architecture permits new components to be “dynamically loaded at run time.” *I.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.”). 6 (“Doubtless, additional plugin types will be introduced by third parties once we have released our code into the public domain.”).

Either alone or as combined with other components identified for the flow by Decasper98, these programs would comprise “a sequence of components for processing a plurality of packets of the message.”

The “data type” of the first packet is analyzed under Implicit’s claims constructions, *e.g.*, by analyzing the data type of the IP option field(s) to determine whether they contain code. Decasper98 alone also analyzes other fields of the first packet which would comprise a “data type,” as explained above. *See* Section V.A.2 (Decasper98 103) at Claim 1(iii).

The sequence is identified “dynamically” because some of its component(s) did not even exist in the system until the first packet arrived. Indeed, it is difficult to imagine how a sequence could be identified any more “dynamically” than by obtaining one or more of the identified components from “the first packet” itself.

Other aspects of this element are discussed above. *See* Section V.A.2 (Decasper98 103) at Claim 1(iii).

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

Claim 1 further recites: “storing an indication of each of the identified components so that the sequence does not need to be re-identified for subsequent packets of the message.” Under Implicit’s apparent claim constructions, Decasper98 in view of Wetherall renders obvious this element.

Wetherall teaches that actively delivered programs “can leave information behind in a node,” and that this information “may be in the form of programs.” Ex. 47 at 34. Wetherall also teaches that such actively delivered programs would be stored and applied to subsequent packets of a message. *Id.* at 33 (“a user . . . could send a customized video transcoding program to a node within the network . . . and request that the node executed that program when processing *their packets.*”) (emphasis added).

This element is further discussed above. *See* Section V.A.2 (Decasper98 103) at Claim 1(iv).

v. “state information”

Claim 1 finally recites: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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 Wetherall renders obvious this element

Wetherall teaches that each actively delivered component can “leave a small amount of associated state at each node along the path it traverses.” Ex. 47 at 34. “Subsequent packets can include code whose execution is dependent on this state.” *Id.*

This element is discussed further above. *See* Section V.A.2 (Decasper98 103) at Claim 1(v).

(b) Claim 4

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

Claim 4 recites: “A method in a computer system for processing a message, the message having a plurality of headers, the method comprising” Under Implicit’s apparent claim constructions, Decasper98 in view of Wetherall renders obvious this element.

In addition to the need for Decasper98 alone to analyze a plurality of headers of the message (*see* Section V.A.1 (Decasper98 102) at Claim 4(i)), Wetherall teaches that one or more IP options headers would need to be analyzed in order to obtain the program(s) they contain. Ex. 47 at 35-36.

ii. dynamically identify a sequence of components

Claim 4 further recites: “analyzing the plurality of headers of a first packet of the message to dynamically identify a sequence of components for processing a plurality of packets of the message such that the output format of the components of the sequence match the input format of the next component in the sequence, wherein analyzing the plurality of headers of the first packet of the message to dynamically identify the sequence of components includes selecting individual components to form the sequence of components after the first packet of the message is received.” Under Implicit’s apparent claim constructions, Decasper98 in view of Wetherall renders obvious this element.

In addition to the need for Decasper98 alone to analyze a plurality of headers of the first packet to identify a sequence of components (*see* Section V.A.1 (Decasper98 102) at Claim 4(ii)), Wetherall teaches that one or more IP options headers would need to be analyzed as well, in order to obtain the program(s) they contain. Ex. 47 at 35-36. Other aspects of this element are discussed above. *See* Claim 1(iii) above.

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

Claim 4 further recites: “storing an indication of each of the identified components so that the sequence does not need to be re-identified for subsequent packets of the message.” Under Implicit’s apparent claim constructions, Decasper98 in view of Wetherall renders obvious this element. *See* Claim 1(iv) above.

iv. “state information”

Claim 4 finally recites: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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 Wetherall renders obvious this element. *See* Claim 1(v) above.

(c) Claim 10

Decasper98 in view of Wetherall renders obvious claim 10. *See* Claim 1 above.

15. Decasper98 in View of Wetherall, RFC 1825, and RFC 1829 Under § 103

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. As applied to the previous combination of Decasper98 in view of Wetherall, RFC 1825 and RFC 1829 further confirm that “a plurality of components” in a sequence would maintain “state information” across packets in the manner recited by claims 1, 4, and 10. *See* Sections V.A.14 (Decasper98+Wetherall) and V.A.3 (Decasper98+RFC 1825+RFC 1829) above.

16. Decasper98 in View of RFC 1825, RFC 1829, RFC 1883, Decasper97, Bellare97, Bellare95, IBM96, Nelson, Fraser, Bellissard, and Wetherall Renders Obvious Claims 1, 4, and 10 Under § 103

If certain aspects recited in claims 1, 4, and 10 of the ‘857 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, Decasper97, Bellare97, Bellare95, IBM96, Nelson, Fraser, Bellissard, and Wetherall in light of the

background knowledge of one of ordinary skill in the art 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.").

Claims 1, 4, and 10 recite elements regarding "state information" as relating to "a plurality of components."

Decasper98 teaches "plugins implementing IPv6 options" and "plugins for IP security." *Id.* at 4. **Decasper97** confirms the obviousness of providing separate IP security plugin components for encryption and authentication. **IBM96** confirms the obviousness of additional plugin components for compression.

RFC 1829 and **Bellare97** confirm the obviousness of employing stateful encryption algorithms which would read on these elements. **Bellare95** confirms the obviousness of employing stateful authentication algorithms which would read on these elements. **Nelson** confirms the obviousness of employing stateful compression algorithms which would read on these elements. **RFC 1883** confirms the obvious of employing a stateful algorithm for implementing IPv6 options which would read on these elements.

Since Decasper98 teaches that its plugin components are selected on the basis of separate, independent filter tables, it was obvious that any two of more of these stateful component types

(encryption, authentication, compression, IPv6 options) would be applied to a particular flow. This was 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”). Moreover, Decasper98 teaches additional stateful components which would read on these “state information” elements, including plugin components for statistics gathering, packet scheduling, and firewall functions—and it was obvious for any of these to be applied to a particular flow as well. *See* Sections V.C.1 (Decasper98 102) and V.C.2 (Decasper98 103) above.

Claims 1, 4, and 10 recite “dynamically identify a sequence of components.”

Decasper98 selects the sequence of plugin components for a flow on the basis of *multiple independent filter tables*. “[E]ven with very few installed filters,” this leads to “exponentially” many valid component sequences—so many, in fact, that it is “infeasible” to even indicate 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”—and including at least to the moment a new flow would begin. *Id.* at 9. This also reads on these “dynamic[.]” claim elements, under Implicit’s apparent claim constructions.

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.

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.

Wetherall teaches dynamically delivering new component(s) in the first packet of a message, which would also read on these "dynamic[]" claim elements. Indeed, it is difficult to imagine how a sequence could be identified any more "dynamically" than by obtaining one or more of the identified components from the first packet itself.

In short, there is no aspect of claims 1, 4, and 10 which was not obvious over the prior art and combinations cited herein.

B. 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 is here presented in a new light relative to the original prosecution of the '857 patent. As noted by statute, 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).

In this case, during prosecution of the ‘857 patent, Mosberger was disclosed by the patentee alongside approximately 20 other references in its Information Disclosure Statement of January 29, 2010, several weeks after the Examiner had declared that the pertinent rejections over the prior art had been overcome. Ex. 40-D (12/11/2009 Final Rejection) at 3 (“Claims 6, 8, 9, 22-24 and 26-28 would be allowable if a terminal disclaimer is filed to overcome the obviousness-type double patenting rejection.”); Ex. 40-F (1/29/2010 IDS). And Mosberger was never discussed or applied by the Examiner at any time during prosecution of the ‘857 patent. Indeed, no evidence exists that the Examiner considered any of the technical teachings of Mosberger to a greater degree than documents are generally considered during a search of PTO file records. *See* MPEP § 2640.

Thus, the anticipatory teachings of Mosberger are here presented in a new light. Moreover, as combined with the various references below (which were not considered by the Examiner), Mosberger is further considered in a new light. Finally, Mosberger is presented in a new light when considered in the context of the *ex parte* reexamination proceedings for the ‘163 patent, including what Implicit has characterized as “*the most important document in the entire case*”—a document that was never made of record in the prior prosecution of the ‘857 patent. *See* Section I above.

I. Mosberger Anticipates Claims 1, 4, and 10 Under § 102(a) and (b)

(a) Claim 1

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

Claim 1 recites: “A method in a computer system for processing packets of a message, 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 employs “a new abstraction called the *path*,” which may be used for processing “a sequence of network packets.” *Id.* at 13 (emphasis in original), 36.

ii. “receiving a packet of the message”

Claim 1 further recites: “receiving a packet of the message and a data type of the message.” Under Implicit’s apparent claim constructions, Mosberger discloses this element.

Mosberger discloses a “Scout Packet Classifier” which “lets Scout pick a path and start processing a packet.” *Id.* at 85-86.

| Without Classifier: | With Classifier: |
|---|---|
| Ethernet processing | Ethernet demux ⇒ it’s an IP packet |
| Ethernet demux ⇒ it’s an IP packet | IP demux ⇒ it’s a UDP packet |
| IP processing | Ethernet processing |
| IP demux ⇒ it’s a UDP packet | IP processing |
| UDP processing | UDP processing |
| ⋮ | ⋮ |

Id. at 86. As shown in the right-hand column of this figure, processing performed “[w]ith” the Scout Packet Classifier includes examining an “IP” packet to determine if it contains, *e.g.*, “UDP” at the layer above. *Id.* As one of ordinary skill in the art would recognize, this

determination is based on the “Protocol” field in the IP packet header, which would comprise a “data type” under Implicit’s apparent claim constructions.²⁵

iii. *“dynamically identify a sequence of components”*

Claim 1 further recites: “analyzing the data type of a first packet of the message to dynamically identify a sequence of components for processing a plurality of packets of the message such that the output format of the components of the sequence match the input format of the next component in the sequence, wherein analyzing the data type of the first packet of the message to dynamically identify the sequence of components includes selecting individual components to form the sequence of components after the first packet of the message is received.” Under Implicit’s apparent claim constructions, Mosberger discloses this element.

Notably, during prosecution of the ‘163 Reexam, the examiner found that Mosberger discloses a similar limitation. *See* Ex. 35-K (‘163 Reexam: Amendment of December 18, 2009) at 3 (portion of ‘163 claim 1 reciting “for the first packet of the message, dynamically identifying a sequence of components for processing the packets of the message . . . wherein dynamically identifying includes selecting individual components to form the sequence of components after the first packet is received”); Ex. 35-L (‘163 Reexam: Advisory Action of January 21, 2010) at 2 (“the proposed amendments, had they been entered, would not have overcome the stated rejections because Mosberger also teaches dynamic selection of components immediately after the first packet is received”). It was only after certain claims were further amended to recite a “non-predefined” sequence that they were allowed. Ex. 35-M (‘163 Reexam: Amendment of February 8, 2010) at 2 (“for the first packet of the message, dynamically identifying a non-

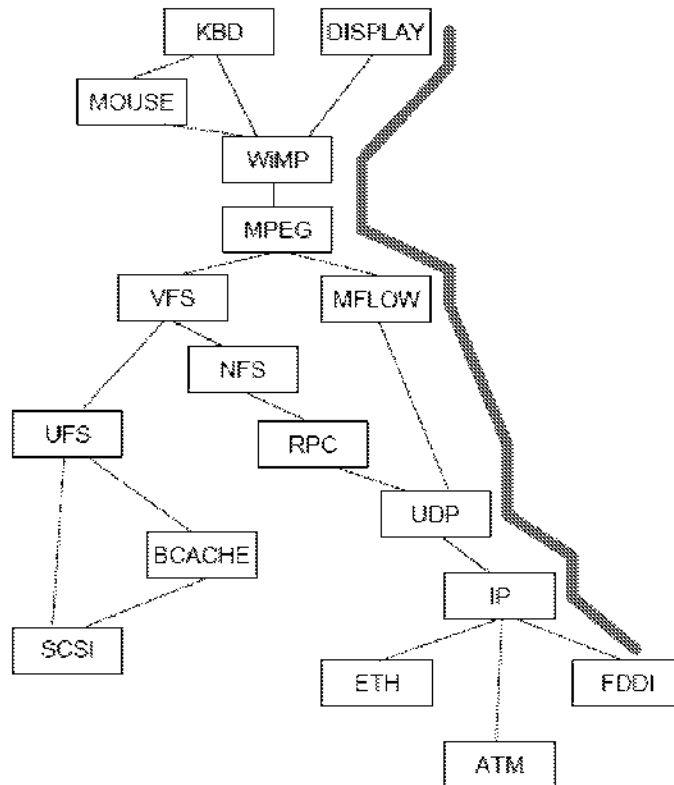
²⁵ *See, e.g.*, Ex. 41 (RFC 791) (IP Specification) (1981) at 14 (“Protocol: 8 bits”; “This field indicates the next level protocol used in the data portion of the internet datagram. The values for various protocols are specified in ‘Assigned Numbers’”). This reference is cited in this context solely to help explain Mosberger. *See* MPEP § 2205.

predefined sequence of components for processing the packets of the message such that the output format of the components of the sequence match the input format of the next component in the sequence, wherein dynamically identifying includes selecting individual components to form the sequence of components after the first packet is received"); Ex. 35-O ('163 Reexam: Examiner Interview Summary of March 5, 2010) at 3 (quoting submission from Applicant regarding the following point of discussion: "The following further amendment to Claim 1 that adds the 'non-predefined' language to the amendment submitted on December 18, 2009:

"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 sequence match the input format of the next component in the sequence, wherein dynamically identifying includes selecting individual components to form the non-predefined sequence of components after the first packet is received" (the language with the single underline was added last December, and the proposed new language to further distinguish over Mosberger is double underlined). It is respectfully submitted that Mosberger does not teach the underlined language because the Mosberger paths are predefined for the reasons set forth in the Patent Owner's prior responses."); Ex. 35-N ('163 Reexam: Notice of Intent to Issue Certificate of March 2, 2010) at 4 ("the amendments to the claims overcome the previous cited prior art in that Mosberger does not dynamically identify sequences of components based only on the first packet with using predefined fields. No other art of record has been found that renders these deficiencies obvious."). Of note, the claims of the '857 patent do not include this "pre-defined" language.

Mosberger teaches that a "path can be visualized as a vertical slice through a layered system" which "typically traverses multiple modules [*i.e.*, components]." Ex. 31 at 13. For

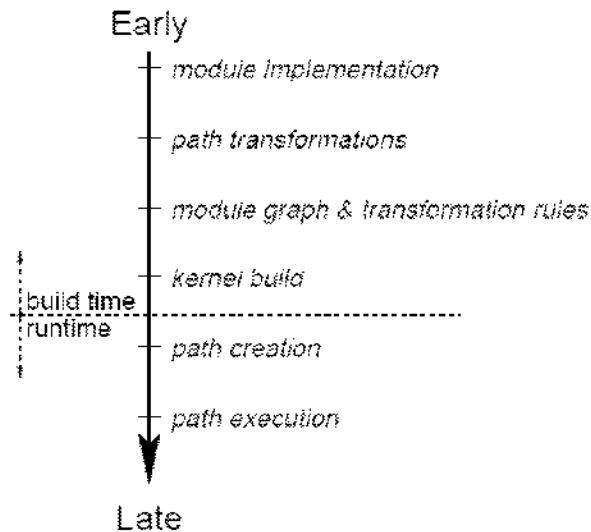
example, the following path is comprised of a sequence of components that obtains video data from packets received at an FDDI network interface, and displays it on a computer monitor:



Id. at 38 (Figure 2.4: “Example Path in Modular System”). *See also id.* at 37-39.

In addition to depicting a particular sequence of components instantiated as a “Path” (green line), the figure immediately above also depicts a data structure which Mosberger refers to as “the module graph.” *Id.* at 41 (“the module graph shown in Figure 2.4”). The module graph defines the set of *permissible* linkages between individual components: “A pair of services can be connected in the module graph only if they are compatible.” *Id.* at 67. Before a pair of components can be “connected in the module graph,” they undergo “compatibility testing” to ensure their input and output “interface[s]” are “compatible.” *Id.* at 67. “A pair of services [components] is considered compatible if the interface *provided* by one service is compatible with the interface *expected* by the other service and vice versa.” *Id.* at 68 (emphasis in original).

This set of permissible linkages between components is defined in the module graph at “build time.”



Id. at 61 (Figure 3.1: “Scout Development Timeline”). Many paths through the module graph are theoretically *possible* given the set of permissible linkages (*e.g.*, in Figure 2.4, from ATM to IP to UDP to RPC to NFS to VFS down to UFS to BCACHE to SCSI). However, at no point does Mosberger go through the exercise of enumerating the various theoretically possible sequences, or “pre-specifying” them into paths.

In fact, Mosberger explicitly rejects that approach. “There are two possible approaches”: (1) “paths are pre-specified (externally)”; or (2) “paths are created (discovered) incrementally.”

Id. at 39. And pre-specification of paths is undesirable 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.* Mosberger explains:

[I]t may seem like pre-specifying paths is the right solution. In such a case, the system would provide a table that translate the properties of the desired path into a sequence of modules that the path needs to traverse to satisfy these properties. Consider the example system shown in Figure 2.4. In this system, there could be a mapping that says that a path to display MPEG-encoded video on

a graphics display must start at module FDDI, go through IP etc., and stops at module DISPLAY. In other words, the mapping would specify the path shown as the bold line in the figure.

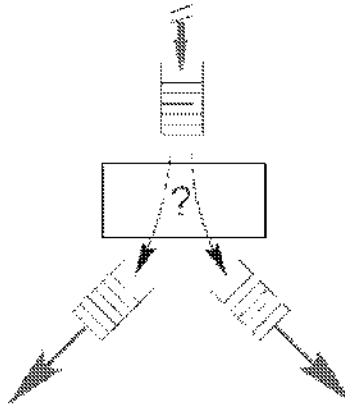
Unfortunately, there are serious problems realizing this approach in practice. Pre-specifying a path often requires detailed knowledge of the internal workings of the modules encountered along a path. For example, whether the path in Figure 2.4 should go from IP to FDDI or to ATM will typically depend on the host that is sending the video and the routing information that is managed by the IP protocol. It certainly is imaginable to embed such detailed knowledge in the part of the system that would manage paths, but it is our contention that a much better solution is to follow the second approach, i.e., to create paths incrementally. With this approach, IP itself can make the decision whether or not the path should extend to ATM or FDDI.

Id. at 40. Thus, the sequence of components to form a path is identified “*incrementally*” at runtime, in response to conditions knowable only then. *Id.*

When the first packet of an incoming message arrives, it may result in one or more “short paths” already created in the system being “extended” to form a complete sequence of components for processing the message. *Id.* at 41 (“Short Paths”: “An artifact of creating paths incrementally is that . . . the created path may be short. For example, with the module graph shown in Figure 2.4, UDP might create a path through IP specifying that *any* remote host is allowed to send packets to this path. In such a case, IP could not make a unique routing decision because packets could arrive through ETH, ATM, or FDDI. The resulting path would be short as it would go from UDP to IP only.”) (emphasis in original), 42 (“Extending Paths”), 44 (the path extension operation “is invoked on the end of an existing path”).

In the course of extending the path in this manner from component to component, various decisions are made “dynamic[ally]”: *e.g.*, “When data arrives from the path above the module, it must make a *dynamic routing decision* to determined [*sic*] whether the data needs to be forwarded to the [short] path at the lower left or the [short] path at the lower right.” *Id.* at 42

(emphasis added). This “dynamic routing decision” is “based on the contents of the data being communicated.” *Id.* at 88.



Id. at 42 (Figure 2.5: “Dynamic Routing Decision”).

“[P]ath 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.” *Id.* at 41. *See also id.* at 47 (“a networking service would typically create a path for its well-known address while the module is being initialized. Then, whenever it receives a new connection request, the service may elect to create a new path to handle the new connection.”).

Because paths in Mosberger are not “pre-specified” but rather identified “incrementally” at runtime in this step-wise manner on the basis of various “dynamic” decisions, the sequence of components for processing the packets of a message is “dynamically identif[ied],” under Implicit’s claim constructions. *Id.* at 39-41.

Claim 1 also recites that the sequence is identified by “analyzing the data type of a first packet of the message.” The identified sequence would clearly depend, *e.g.*, on whether the Protocol field of the first packet indicates TCP or UDP, and this would comprise a “data type” under Implicit’s apparent claim constructions.

Claim 1 also recites “such that the output format of the components of the sequence match the input format of the next component in the sequence.” The module graph is consulted while performing the incremental identification of components for a message, to assure the linkages are compatible. The module graph defines the set of *permissible* linkages between components: before a pair of components can even be “connected in the module graph,” they undergo “compatibility testing” to ensure their input and output “interface[s]” are “compatible.” *Id.* at 67. “A pair of services [components] is considered compatible if the interface *provided* by one service is compatible with the interface *expected* by the other service and vice versa.” *Id.* at 68 (emphasis in original).

Claim 1 also recites: “selecting individual components to form the sequence of components after the first packet of the message is received.” As explained above, various individual components are selected in the course of incrementally identifying a sequence. *See id.* at 39-44.

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

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

Notably, during prosecution of the ‘163 Reexam, the examiner found that Mosberger anticipates an identically-worded limitation. *See* Ex. 35-H (‘163 Reexam: Amendment of September 1, 2009) at 3 (portion of ‘163 claim 1 reciting “storing an indication of each of the

identified components so that the sequence does not need to be re-identified for subsequent packets of the message”). The examiner observed: “Since paths are fixed for the lifetime of the path (see p. 54, section 2.4.1.2), any thread created for that path must inherently be able to refer to stored information related to that path in order to continually process messages.”). Ex. 35-J (‘163 Reexam: Final Office Action of December 4, 2009) at 7.

And indeed, Mosberger discloses that “the sequence of modules being traversed is known and fixed for the lifetime of a path.” Ex. 31 at 54 (section 2.4.1.2). 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 finally recites: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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 this “state information” element.

Notably, during prosecution of the ‘163 Reexam, the examiner found that Mosberger anticipates a similar limitation. *See* Ex. 35-H (‘163 Reexam: Amendment of September 1, 2009) at 5 (portion of ‘163 claim 15 reciting “for each packet of the message, performing the processing of the identified 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.”). The examiner observed: “Since Mosberger discloses that packets may be fragmented, so a component may need multiple packets before processing is possible (see pp. 86-88), the state of a component’s processing of a fragment must be available for the subsequent processing of the next fragment in that same component.” Ex. 35-J (‘163 Reexam: Final Office Action of December 4, 2009) at 9.

And indeed, Mosberger teaches that a layer 3 IP component would maintain state information across packets in the manner recited by claim 1 of the '857 patent, in order to perform layer 3 packet defragmentation. Ex. 31 (Mosberger) at 87 (“Once IP receives the fragment . . . it will buffer the fragment until the entire datagram has been reassembled.”).

Additionally, Mosberger teaches that a layer 4 TCP component would maintain (and adjust the size of) a “flow-control window” for the processing of its “bytestream.” *Id.* at 79. As one of ordinary skill in the art would appreciate, performing flow control on a TCP bytestream would entail maintaining state information across packets (*e.g.*, sequence numbers) in the manner recited by claim 1.²⁶

Moreover, 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. 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.

²⁶ *See, e.g.*, Ex. 9 (RFC 793) (“Transmission Control Protocol” [TCP] Specification) (1981) 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 Mosberger. *See* MPEP § 2205.

(b) Claim 4

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

Claim 4 recites: “A method in a computer system for processing a message, the message having a plurality of headers, the method comprising . . .” Under Implicit’s apparent claim constructions, Mosberger discloses this element.

As explained above, Mosberger discloses a “Scout Packet Classifier” which “lets Scout pick a path and start processing a packet.” Ex. 31 at 85-86. This includes analysis of an “Ethernet” header to determine “it’s an IP packet,” and analysis of an “IP” header to determine “it’s a UDP packet.” *See id.* at 86.

ii. “dynamically identify a sequence”

Claim 4 further recites: “analyzing the plurality of headers of a first packet of the message to dynamically identify a sequence of components for processing a plurality of packets of the message such that the output format of the components of the sequence match the input format of the next component in the sequence, wherein analyzing the plurality of headers of the first packet of the message to dynamically identify the sequence of components includes selecting individual components to form the sequence of components after the first packet of the message is received.” Under Implicit’s apparent claim constructions, Mosberger discloses this element.

Regarding the limitation “*analyzing the plurality of headers* of a first packet of the message to . . . identify a sequence of components,” Mosberger discloses a “Scout Packet Classifier” which “lets Scout pick a path and start processing a packet.” Ex. 31 at 85-86. This includes analysis of an “Ethernet” header to determine “it’s an IP packet,” and analysis of an “IP” header to determine “it’s a UDP packet.” *See id.* at 86.

Other aspects of this claim element are discussed above. *See* Claim 1(iii) above.

iii. Other claim elements

The remaining elements of claim 4 are also disclosed by Mosberger. *See* Claim 1 above.

(c) Claim 10

Claim 10 recites in pertinent part: “A computer readable storage medium, other than a data transmission medium, containing instructions for processing packets of a message, the instructions comprising at least one computer-executable module configured to” Under Implicit’s apparent claim constructions, Mosberger discloses this element.

Mosberger teaches a software-based system, and one of ordinary skill would recognize this software would be dynamically loaded from a “computer readable storage medium, other than a data transmission medium”: *e.g.*, from a hard disk in the device. *E.g.*, Ex. 30 at 20 (“entire system software”). Other aspects of this claim element are discussed above. *See* Claim 1(i) above.

The remaining elements of claim 10 are also disclosed by Mosberger. *See* Claim 1 above.

2. Mosberger Renders Obvious Claims 1, 4, and 10 Under § 103

If certain aspects recited in claims 1, 4, and 10 of the ‘857 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.

(a) Claim 1

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

Claim 1 recites: “A method in a computer system for processing packets of a message, the method comprising” Under Implicit’s apparent claim constructions, Mosberger discloses this element. *See* Section V.B.1 (Mosberger 102) at Claim 1(i) above.

ii. “receiving a packet of the message”

Claim 1 further recites: “receiving a packet of the message and a data type of the message.” Under Implicit’s apparent claim constructions, Mosberger discloses this element. *See* Section V.B.1 (Mosberger 102) at Claim 1(ii) above.

iii. “dynamically identify a sequence of components”

Claim 1 further recites: “analyzing the data type of a first packet of the message to dynamically identify a sequence of components for processing a plurality of packets of the message such that the output format of the components of the sequence match the input format of the next component in the sequence, wherein analyzing the data type of the first packet of the message to dynamically identify the sequence of components includes selecting individual components to form the sequence of components after the first packet of the message is received.” Under Implicit’s apparent claim constructions, Mosberger renders obvious this element.

As explained above, Mosberger discloses this element. *See* Section V.B.1 (Mosberger 102) at Claim 1(iii) above. However, even if Mosberger is deemed not to have an express disclosure of the “dynamically identify” 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.* (emphasis added). 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.*²⁷ And, of course, the claims of the ‘857 patent contain no limitation directed to any such “security issue”; in other words, even an insecure implementation of “dynamic loading” would satisfy the “dynamically identify” limitation of the ‘857 patent.

Mosberger earlier explains how an edge of the module graph would be reached: “path creation is initiated at the module that is to form one end of the path This [routing decision] process repeats itself until there is no next module (i.e., the edge of the module graph has been reached).” *Id.* at 41. By explaining that the proposed “dynamic module-loading facility” would address the lack of an ability “to extend the [module] graph at runtime,” Mosberger clearly communicates that these new, dynamically-loaded modules could permit a path to be dynamically extended when the edge of the module graph had been reached. *See id.* at 71. To pick a simple example, one of ordinary skill in the art would understand the “Protocol” field of a layer 3 IP packet indicates the layer 4 protocol above. While this field commonly indicates TCP or UDP, one of ordinary skill in the art would recognize it can call for many other layer 4

²⁷ In connection with the concurrent litigation, a deposition of the author of the Mosberger reference, David Mosberger, was conducted on September 16, 2011. *See Ex. 50* (Mosberger Depo). Mr. Mosberger explained: “in the architecture we clearly tried to design it to accommodate *dynamic extensibility*, and *that’s what the statement on page 71 refers to.*” *Id.* at 92:10-13. Asked, “How would the dynamic module loading facility work exactly?,” Mr. Mosberger explained: “Well, it would be akin to basically a dynamically library loader as is used on any modern operating system like Linux, Windows, UNIX, Mac OS. *So that’s standard technology basically*, and then the only component that would be different – so a module in that case you would most likely realize is a separate what’s often called a DLL or a dynamically shared library.” *Ex. 50* at 182:7-18. Mr. Mosberger also remarked that he “would very much have liked to go in [the] direction [of writing more about the dynamic loading facility], but it wasn’t considered to be a topic of research because it’s *basically well-known technologies*” *Id.* at 185:9-21.

protocols as well.²⁸ Hence, it would be obvious for a “dynamic module-loading facility” as proposed by Mosberger to dynamically load a suitable protocol module when, *e.g.*, the first packet of a message is received which indicates a layer 4 protocol not currently configured in the module graph. *See id.* at 38 (showing module graph supporting “UDP,” but not TCP or other possibilities), 41 (“the module graph shown in Figure 2.4”). This simple example is merely illustrative, and many other dynamic extensions to the module graph are clearly possible: *e.g.*, dynamically loading an application module based on the value of a layer 4 port number.²⁹

A system equipped with such a “dynamic module-loading facility” would thus clearly read on this claim element: “analyzing the data type of a first packet of the message to dynamically identify a sequence of components for processing a plurality of packets of the message such that the output format of the components of the sequence match the input format of the next component in the sequence.”

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.” Ex. 31 at 71. 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

²⁸ *See, e.g.*, Ex. 42 (RFC 1700) (“Assigned Numbers”) (1994) at 8 (“In the Internet Protocol (IP) . . . there is a field, called Protocol, to identify the next level protocol”; possible values include “Stream . . . Network Voice Protocol . . . Reliable Data Protocol,” as well as TCP and UDP). This reference is cited in this context solely to help explain Mosberger. *See* MPEP § 2205.

²⁹ *See, e.g.*, Ex. 42 (RFC 1700) at 16-18 (“WELL KNOWN PORT NUMBERS” including “Telnet . . . Simple Mail Transfer . . . Graphics . . . XNS Mail” and so on. This reference is cited in this context solely to help explain Mosberger. *See* MPEP § 2205.

“virtual machine” to be used to permit code to be dynamically “downloaded and executed inside Scout.” *Id.*

Claim 1 also recites “such that the output format of the components of the sequence match the input format of the next component in the sequence.” The dynamically loaded module is clearly loaded for this very purpose: that its input format (*e.g.*, a particular layer 3 or 4 protocol) would match the output format of the module at the end of the current, partially identified sequence.

Claim 1 also recites “selecting individual components to form the sequence of components after the first packet of the message is received.” The dynamically loaded module is clearly an individual module, which is incrementally connected to a sequence of other module(s) which had been individually selected.

Thus, the express suggestion of Mosberger to add a “dynamic module-loading facility to Scout” renders obvious (or even further obvious) all aspects of this claim element.

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

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

Because Mosberger teaches that “the sequence of modules being traversed is known and fixed for the lifetime of a path,” it was certainly at least obvious that the sequence of modules for any given path would be stored as recited in this element. Ex. 31 at 54.

v. “state information”

Claim 1 finally recites: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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 renders obvious this “state information” element.

It was certainly at least obvious for the layer 3 IP and layer 4 TCP components discussed above to store state information across packets in the manner recited by this element, in order to function as described. *See* Section V.B.1 (Mosberger 102) at Claim 1(v) above.

(b) Claim 4

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

Claim 4 recites: “A method in a computer system for processing a message, the message having a plurality of headers, the method comprising” Under Implicit’s apparent claim constructions, Mosberger discloses this element. Section V.B.1 (Mosberger 102) at Claim 4(i) above.

ii. “dynamically identify a sequence”

Claim 4 further recites: “analyzing the plurality of headers of a first packet of the message to dynamically identify a sequence of components for processing a plurality of packets of the message such that the output format of the components of the sequence match the input format of the next component in the sequence, wherein analyzing the plurality of headers of the first packet of the message to dynamically identify the sequence of components includes selecting individual components to form the sequence of components after the first packet of the message is received.” Under Implicit’s apparent claim constructions, Mosberger renders obvious this element.

Regarding the limitation “*analyzing the plurality of headers* of a first packet of the message to . . . identify a sequence of components,” as part of the “dynamic module-loading facility” proposed by Mosberger, it was obvious that components would be dynamically loaded based on analysis of information in, *e.g.*, a layer 3 or layer 4 header. *See* Claim 1(iii) above.

This is in addition to the other analysis of packet headers already performed by Mosberger. *See* Section V.B.1 (Mosberger 102) at Claim 4(ii) above. Other aspects of this element are discussed above. *See* Claim 1(iii) above.

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

Claim 4 further recites: “storing an indication of each of the identified components so that the sequence does not need to be re-identified for subsequent packets of the message.” Under Implicit’s apparent claim constructions, Mosberger renders obvious this element. *See* Claim 1(iv) above.

iv. “state information”

Claim 4 finally recites: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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 renders obvious this element. *See* Claim 1(v) above.

(c) Claim 10

Mosberger renders obvious claim 10. *See* Claim 1 above.

3. Mosberger in View of HotLava Renders Obvious Claims 1, 4, and 10 Under § 103

The article “Implementing Communication Protocols in Java” (Exhibit 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 ‘857 patent.

If certain aspects recited in claims 1, 4, and 10 of the ‘857 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 in light of the background knowledge of one of ordinary skill in the art, 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.

(a) Claim 1

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

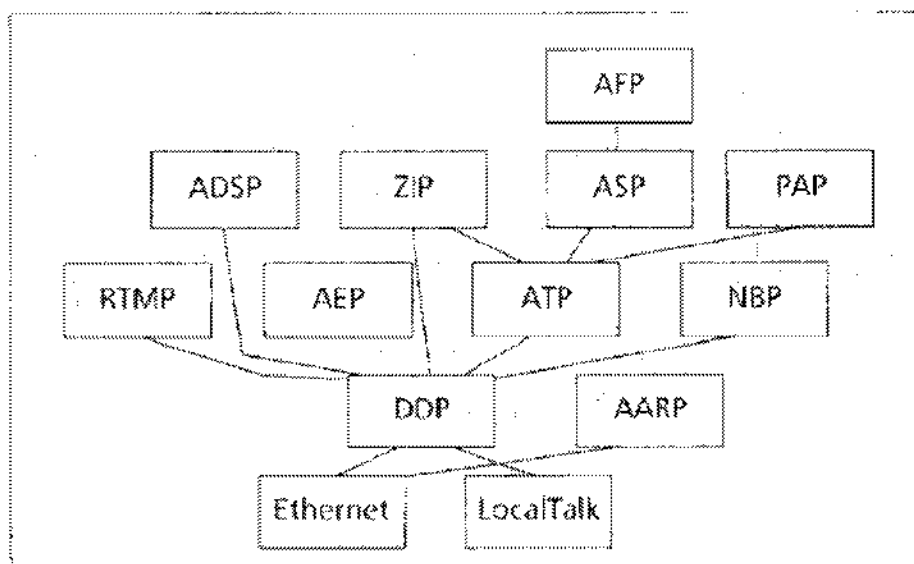
Claim 1 recites: “A method in a computer system for processing packets of a message, the method comprising” Under Implicit’s apparent claim constructions, Mosberger in view of HotLava renders obvious this element. *See, e.g.*, Ex. 32 at 93-94, 95, 96, 97.

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. “receiving a packet of the message”

Claim 1 further recites: “receiving a packet of the message and a data type of the message.” Under Implicit’s apparent claim constructions, Mosberger in view of HotLava renders obvious this element.

HotLava explains: “For each incoming . . . message a protocol processes, it must identify the correct next protocol . . . and forward the message to it for processing.” *Id.* at 94.



■ **Figure 1.** *Example protocol graph (AppleTalk).*

Id. at 94. As one of ordinary skill in the art would understand, the “correct next protocol” is generally ascertained by examining a field in the previous layer’s header which would comprise a “data type” under Implicit’s apparent claim constructions. *See id.* at 94 (“each individual protocol . . . removes and interprets headers from *incoming* messages received from the network”) (emphasis in original).

iii. “dynamically identify a sequence of components”

Claim 1 further recites: “analyzing the data type of a first packet of the message to dynamically identify a sequence of components for processing a plurality of packets of the message such that the output format of the components of the sequence match the input format of the next component in the sequence, wherein analyzing the data type of the first packet of the message to dynamically identify the sequence of components includes selecting individual components to form the sequence of components after the first packet of the message 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 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.

HotLava also 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 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: "for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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 4

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

Claim 4 recites: “A method in a computer system for processing a message, the message having a plurality of headers, the method comprising . . .” Under Implicit’s apparent claim constructions, Mosberger in view of HotLava renders obvious this element.

HotLava explains: “For each incoming . . . message a protocol processes, it must identify the correct next protocol . . . and forward the message to it for processing.” *Id.* at 94. As one of ordinary skill in the art would understand, in order to do so identify the correct sequence of protocols, a plurality of headers would be analyzed. *See id.* (“each individual protocol [component] . . . removes and interprets headers from *incoming* messages received from the network”) (emphasis in original).

ii. “dynamically identify a sequence”

Claim 4 further recites: “analyzing the plurality of headers of a first packet of the message to dynamically identify a sequence of components for processing a plurality of packets of the message such that the output format of the components of the sequence match the input format of the next component in the sequence, wherein analyzing the plurality of headers of the first packet of the message to dynamically identify the sequence of components includes selecting individual components to form the sequence of components after the first packet of the message is received.” Under Implicit’s apparent claim constructions, Mosberger in view of HotLava renders obvious this element.

Regarding the limitation “*analyzing the plurality of headers* of a first packet of the message to . . . identify a sequence of components,” HotLava explains: “For each incoming . . . message a protocol processes, it must identify the correct next protocol . . . and forward the message to it for processing.” *Id.* at 94. As one of ordinary skill in the art would understand, in

order to do so identify the correct sequence of protocols, a plurality of headers would be analyzed. *See id.* (“each individual protocol [component] . . . removes and interprets headers from *incoming* messages received from the network”) (emphasis in original).

Other aspects of this element are discussed above. *See* Claim 1(iii) above.

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

Claim 4 further recites: “storing an indication of each of the identified components so that the 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* Claim 1(iv) above.

iv. “state information”

Claim 4 finally recites: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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* Claim 1(v) above.

(c) Claim 10

Mosberger in view of HotLava renders obvious claim 10. *See* Claim 1 above.

4. HotLava Anticipates Claims 1, 4, and 10 Under § 102(a) and (b)

The HotLava reference not only renders the claims of the ‘857 patent when considered in combination with Mosberger, but HotLava also independently and standing alone discloses each and every element of claims 1, 4, and 10. 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.

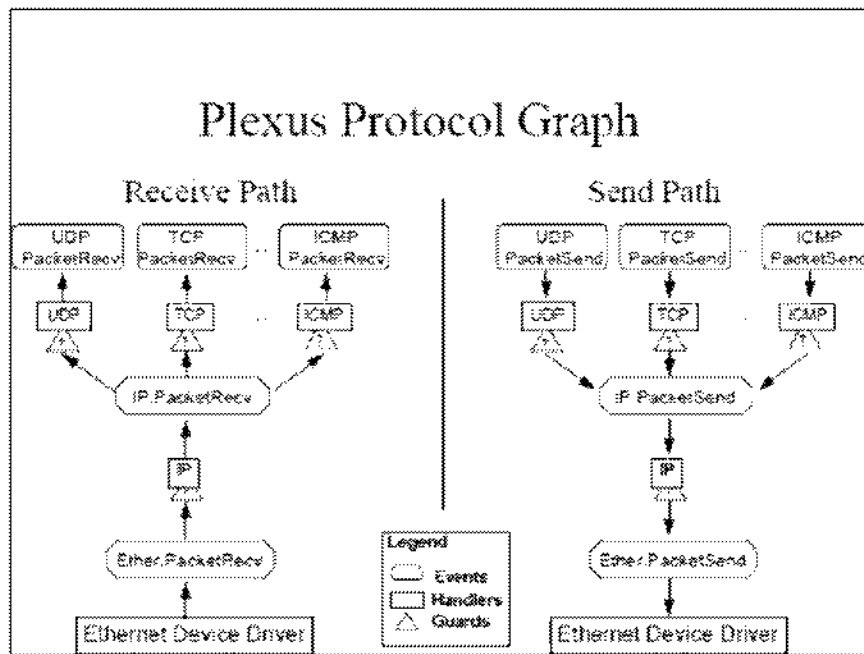
5. Mosberger in View of Plexus Renders Obvious Claims 1, 4, and 10 Under § 103

The article “An Extensible Protocol Architecture for Application-Specific Networking” (Exhibit 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 ‘857 patent.

If certain aspects recited in claims 1, 4, and 10 of the ‘857 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 in light of the background knowledge of one of ordinary skill in the art, 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, 4, and 10, the combination of Mosberger with Plexus clearly makes up for any such perceived deficiency.

6. Mosberger in View of ComScript Renders Obvious Claims 1, 4, and 10 Under § 103

The article “ComScript: An Environment for the Implementation of Protocol Stacks and their Dynamic Reconfiguration” (Exhibit 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 ‘857 patent.

If certain aspects recited in claims 1, 4, and 10 of the ‘857 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 in light of the background knowledge of one of ordinary skill in the art, 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.

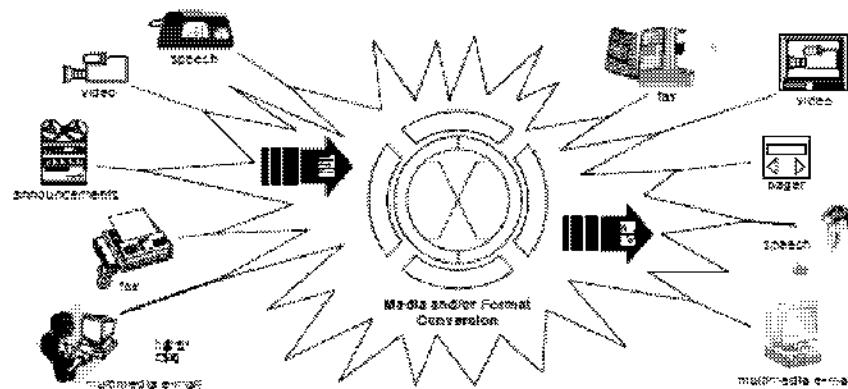
Id. at 6-7; Fig. 10; see also Figs. 7-9 (illustrating how to add or remove a “module” from a stack; “the number of configurable entities is unlimited”).

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, 4, and 10, the combination of Mosberger with ComScript clearly makes up for any such perceived deficiency.

C. Pfeifer96 (Exhibit 3)

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. 3 at 105, 117.



Id. at 118 (Figure 9: “Priorized media conversion in the iPCSS”).³⁰ None of these iPCSS references (including Pfeifer96) were considered during prosecution of the ‘857 patent.

³⁰ Note the figures from Pfeifer96 in this Request are drawn from Ex. 3-B, but aside from being in color, they are identical to the figures in Ex. 3.

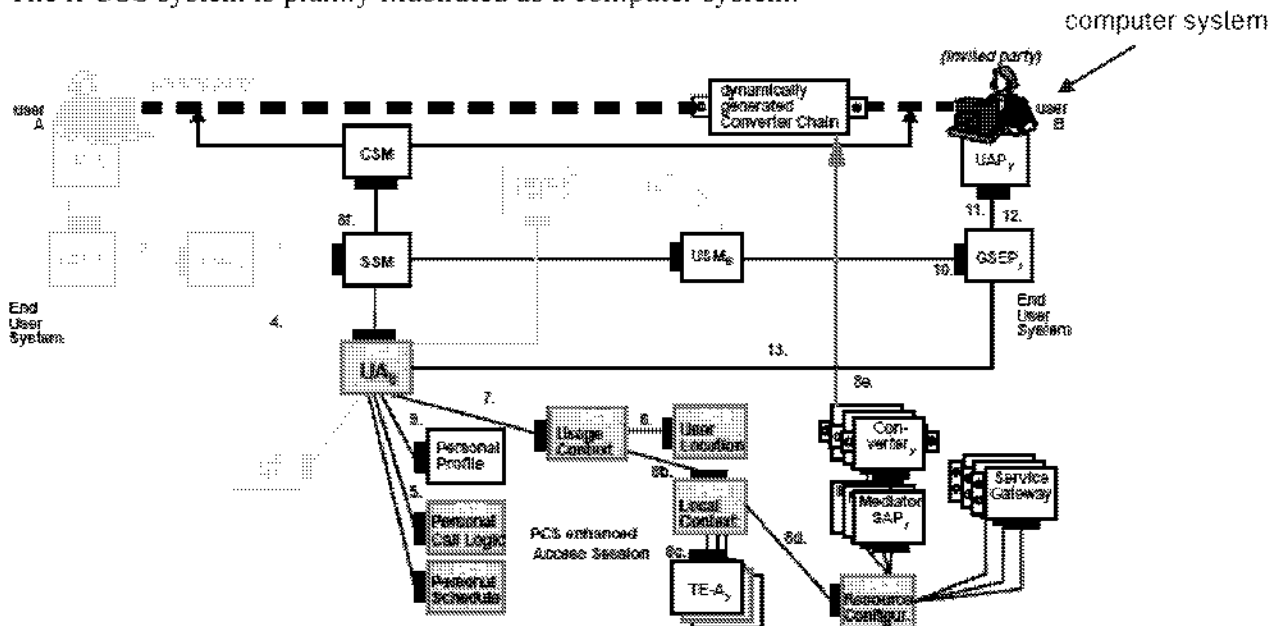
1. Pfeifer96 Anticipates Claims 1, 4, and 10 Under § 102(a), (b)

(a) Claim 1

i. "A method . . . for processing packets of a message"

Claim 1 recites: "A method in a computer system for processing packets of a message, the method comprising . . ." Under Implicit's apparent claim constructions, Pfeifer96 discloses this element.

The iPCSS system is plainly illustrated as a computer system:

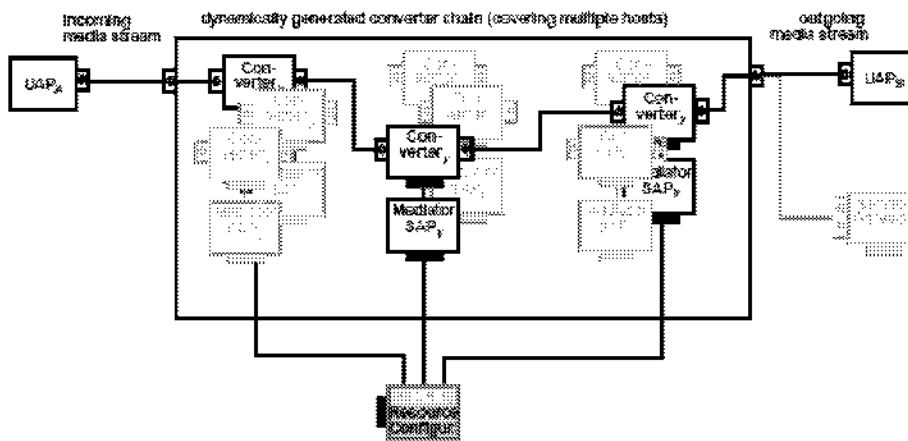


Id. at 122 (Figure 11: "Components of the PCS-enhanced TINA³¹ Access Session," showing a "dynamically generated Converter Chain" connecting the two parties). Pfeifer96 teaches a "system/platform" implementing the "iPCSS architecture," wherein communication between two parties over "fixed" and/or "wireless networks" is mediated by a "chain of converters" which is "dynamically generated." Ex. 3 at 119, 105, 114, 124. Each "converter" in this chain may

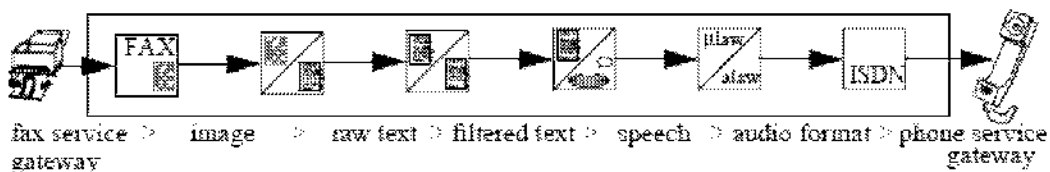
³¹ The "Telecommunications Information Networking Architecture (TINA)" was "developed by a consortium which is currently the focus of worldwide attention." *Id.* at 113. iPCSS is an enhancement to this architecture, adding "Personal Communications Support." *Id.* at 105.

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 recites the method is “for processing . . . a message.” 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.



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 message 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 will be explained in more detail below, this use of “chains of converters” which transform messages 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) “processing . . . a message.”

Claim 1 recites the method is “for processing packets of a message.” Pfeifer96 discloses this in several manners. 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.³² *Id.* at 109, 111. A voice call over an ISDN network, as illustrated in Pfeifer96 (*see id.*) would thus comprise “packets of a message.” In addition, Pfeifer96 discloses application in a “TCP/IP-based” environment, which one of ordinary skill in the art would again understand is inherently packetized. *Id.* at 114 (“controller framework . . . controls . . . data flow via fire redirection, pipes or TCP/IP-based services”).³³ Thus, for example, “reception and delivery of multimedia e-mail” in a TCP/IP-based environment would comprise “packets of a message.” *Id.* at 118, 126. Other packet-based embodiments are also disclosed. *See, e.g., id.* at 105 (“mobility of the user in fixed networks and wireless networks”) (one of ordinary skill recognizing that wireless networks are inherently packetized), 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”), 126 (“sending and reception of . . . faxes”) (one of ordinary skill recognizing that fax machines are commonly positioned on inherently packetized networks such as ISDN), 126 (“multimedia conferencing”).

³² *See, e.g.,* Ex. 4 (ISDN98) (“ISDN Primary Rate User-Network Interface Specification”) (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.

³³ *See, e.g.,* Ex. 41 (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.

In short, Pfeifer96 teaches a “universal platform” meant to provide “universal connectivity” over both “fixed and wireless networks,” and this would include “a computer system for processing packets of a message.” *See id.* at 105, 120, 117-18.

ii. “receiving a packet of the message”

Claim 1 further recites: “receiving a packet of the message and a data type of the message.” Under Implicit’s apparent claim constructions, Pfeifer96 discloses this element.

Packets of incoming messages of various data types are processed by Pfeifer96, including, *e.g.*: voice calls; “multimedia e-mail”; “faxes”; “multimedia conferencing”; and so on. *E.g.*, Ex. 3 at 105, 118, 126; Claim 1(i) above. As explained in the following section, an incoming data type is crucial for determining the sequence of conversions needed to deliver the message to its intended recipient at his or her current location.

iii. “dynamically identify a sequence of components”

Claim 1 further recites: “analyzing the data type of a first packet of the message to dynamically identify a sequence of components for processing a plurality of packets of the message such that the output format of the components of the sequence match the input format of the next component in the sequence, wherein analyzing the data type of the first packet of the message to dynamically identify the sequence of components includes selecting individual components to form the sequence of components after the first packet of the message is received.” Under Implicit’s apparent claim constructions, Pfeifer96 discloses this element.

Pfeifer96 explains: “The iPCSS architecture . . . aim[s] . . . to increase the nomadic user’s reachability by introducing . . . the dynamic selection of terminals,” thereby “allow[ing] people to make use of any kind of terminal located at their whereabouts for obtaining access to their service.” Ex. 3 at 122, 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

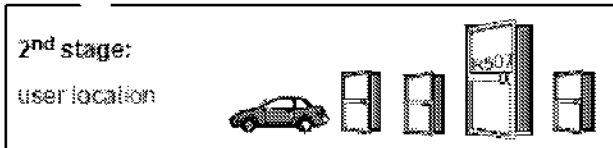
specific media conversions that will be needed to achieve a connection to the user until the first packet of the message to that user has been received by the iPCSS. *See id.* at 119.

Specifically, when an incoming connection is initiated (*i.e.*, when the first packet of the message has been received by iPCSS), the system must determine: (1) a data type (*e.g.*, the “medium”) for the incoming message (*e.g.*, is it a voice call, a fax, or an email?); and (2) its intended “recipient” (*i.e.*, the “called party”). *Id.* at 120, 126-27. Once these are known, the iPCSS can then determine: (3) the called party’s current “location”; and (4) “the set of all access devices in” the called party’s “current vicinity” (*e.g.*, is there an available telephone, fax machine, or computer?). *Id.* at 120, 126-27. Once these are known, the iPCSS can then determine if “*a conversion into another medium*” would be necessary to complete the connection (*e.g.*, if there is not a fax machine in the vicinity, could the fax be read to the user over the telephone using a Text-to-Speech engine?). *Id.* at 120 (emphasis in original), 126-27, 111. If a conversion is necessary, the iPCSS will then “**dynamically**” generate “an appropriate converter chain” to accomplish the conversion, and the connection can proceed. *Id.* at 127 (emphasis added), 120.

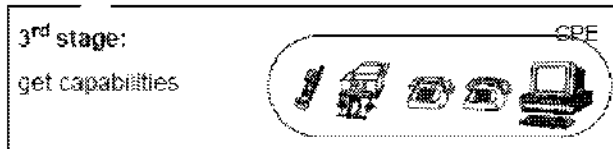
This is why Pfeifer96 teaches a multi-stage call connection procedure, wherein the connection request is received in the “1st stage,” the called user’s “location” is ascertained in the “2nd stage,” the set of available access devices (“terminal equipment”) at that location and their “capabilities” are 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).

personal ID

from basic call processing/
connection control



same selection process as above



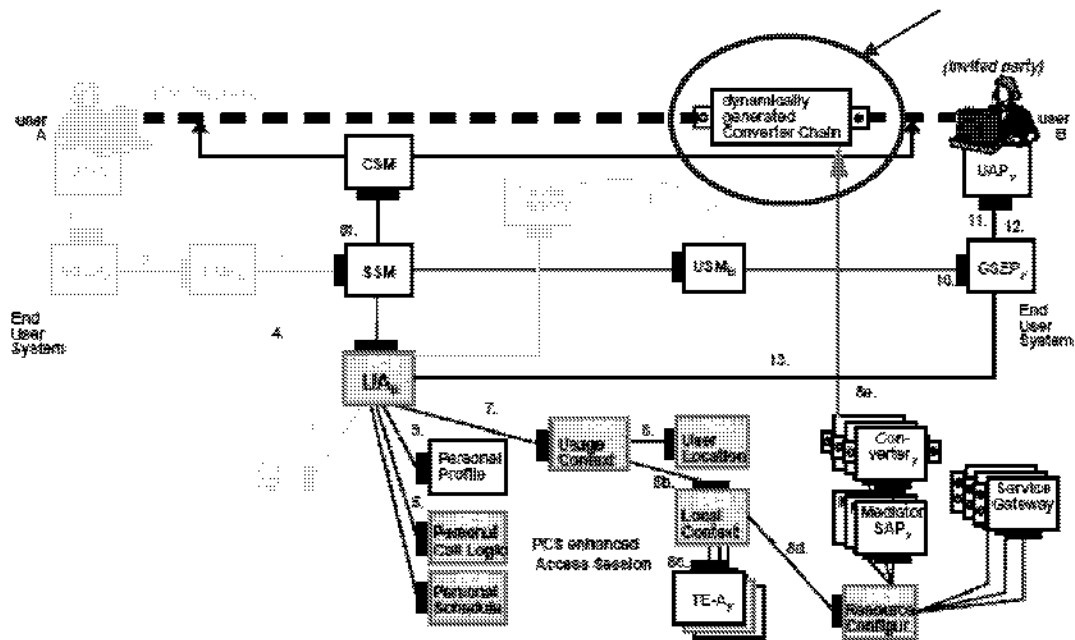
terminal ID

to basic call processing/
connection control



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), 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:



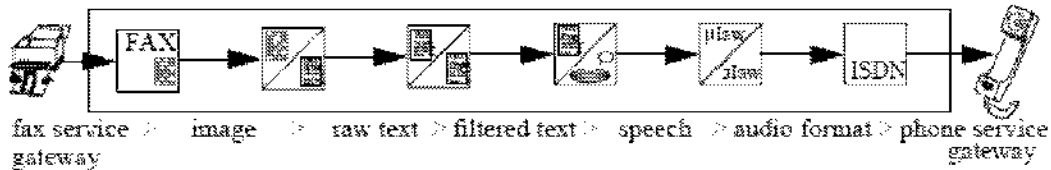
Id. at 122 (Figure 11: “Components of the PCS-enhanced TINA Access Session,” showing the connection request at step “1,” and the “**dynamically generated Converter Chain**” at end of step “8e”) (emphasis added).

Claim 1 recites “analyzing the data type of a first packet of the message.” As explained above, when a connection begins, the iPCSS must determine a data type (*e.g.*, “medium”) for the message in order to determine the necessary sequence of conversions (if any). *Id.* at 119-20, 126-27.

Claim 1 further recites “to dynamically identify a sequence of components for processing a plurality of packets of the message.”

As explained above, if conversion is needed, the iPCSS will then “dynamically generate[]” an “appropriate converter chain.” *Id.* at 127, 124. More specifically, after the first packet of the message has arrived (and in the “4th stage” of the connection process), a component called the “Resource Configurator” systematically evaluates various possible “chains of multiple converters” that could adapt the message to one of the available destination devices. *Id.* at 119-20, 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”).

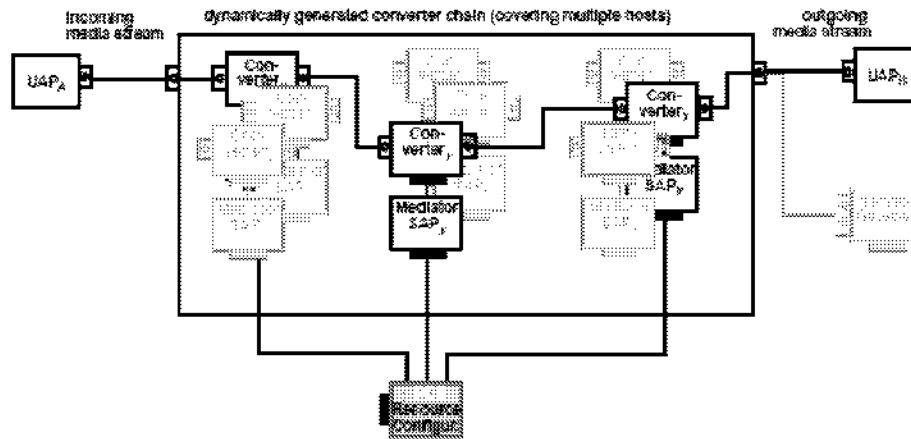
While finding a *workable* chain is certainly an aspect of the analysis performed by the Resource Configurator, the Configurator goes even further and attempts to find an *optimal* chain. It does so by performing a complex “Quality of Service (QoS)” analysis which compares multiple possible “chains” which could be configured to accomplish the conversion, in order to “select the chain most appropriate for the desired task.” *Id.* at 124, 115. Factors included in this QoS comparison of possible chains include: (1) “Intelligibility” (*e.g.*, it would be preferable to avoid chains employing Optical Character Recognition or Speech Recognition components, as opposed to chains which would allow the recipient to experience the message in more of its original form); (2) “Quality degradation due to entropy reduction” (*e.g.*, it would be preferable to avoid chains delivering “24 bit” image data to devices with an “8 bit screen”); (3) “Quality degradation due to lossy compression/decompression” (*e.g.*, it would be preferable to avoid chains that would perform “multiple lossy compression and decompression”); and (4) “Delay” (*e.g.*, it would be preferable to avoid chains employing a Text-to-Speech component, which

“might wait for the end of a sentence” to determine “correct prosody”). *Id.* at 115-16, 124. As summarized by Pfeifer96: “Comparing different possibilities of concatenating converters for a specific task requires a complex evaluation of the quality parameters involved, **performed at runtime.**” *Id.* at 116 (emphasis added).

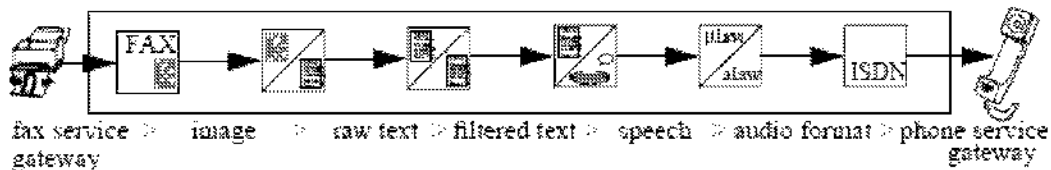
This complex generation and comparison of possible candidate chains is performed at runtime, after the first packet of the incoming message has arrived, because all of the candidate chains are custom-tailored to convert *from* the specific data type of the incoming message *to* a data type which can be received by one of the devices in the recipient’s current vicinity. *Id.* at 115-16, 119-20, 124. The incoming data type may change from message to message, and the recipient’s location (and hence the devices in that vicinity) may change from moment to moment as well. Thus, when the most “appropriate converter chain” is finally selected from among these candidates, it has been (as described by Pfeifer96) “**dynamically generated**”—on demand, as needed. *Id.* at 124, 127.

Under Implicit’s apparent claims constructions, this iPCSS technique would comprise (as recited by claim 1) “dynamically identify[ing] a sequence of components for processing a plurality of packets of the message.”

Claim 1 further recites “such that the output format of the components of the sequence match the input format of the next component in the sequence.” 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:

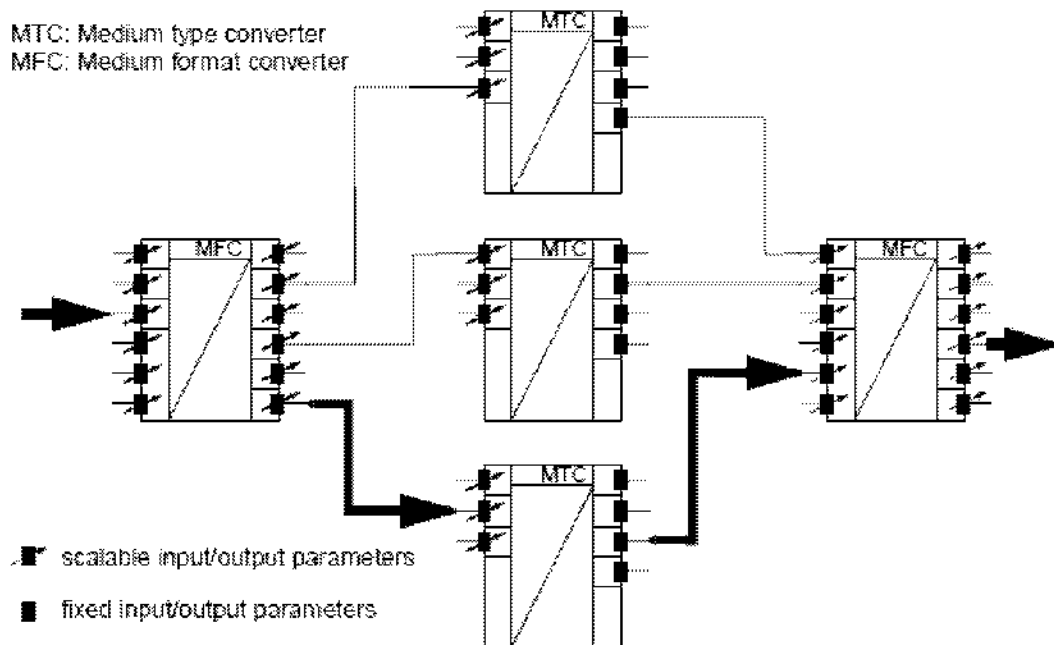


Id. at 111 (Figure 6: “Converter chain: fax reception, conversion to text and speech, telephone delivery”). As apparent from the above example (where a fax is converted through a series of five intermediate formats to a telephone call), the converters are arranged “such that the output format of the components of the sequence match the input format of the next component in the sequence.”

Pfeifer96 explains in more detail: “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 106.



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. The format is altered in either case, because 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. And in either case, it is necessary to arrange the converters “such that the output format of the components . . . match the input format of the next component.” 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):



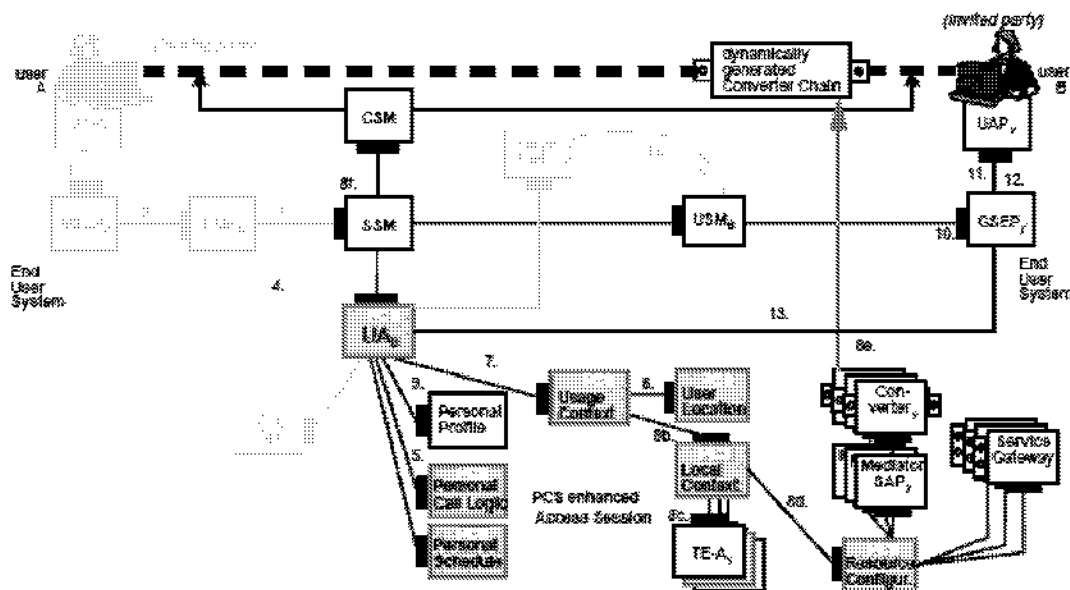
Id. at 108 (Figure 4: “Medium type conversion with format adaptation”). Thus, under Implicit’s apparent claims constructions, Pfeifer96 discloses that (as recited by claim 1) “the output format of the components of the sequence match the input format of the next component in the sequence.”

Claim 1 further recites “analyzing the data type of the first packet of the message to dynamically identify the sequence of components includes selecting individual components to form the sequence of components after the first packet of the message is received.” As explained above, Pfeifer96 teaches “the controlled combination (concatenation) of various [individual] converters,” and these individual components are dynamically assembled into many possible candidate chains after the first packet of the message is received. *Id.* at 105, 115-16, 119-20, 124. Thus, the candidate chain which is ultimately chosen (as well as all the candidate chains which were not) is formed by selecting individual components after the first packet of the message was received.

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

Claim 1 further recites: “storing an indication of each of the identified components so that the 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.” Ex. 3 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* Claim 1(iii) 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.

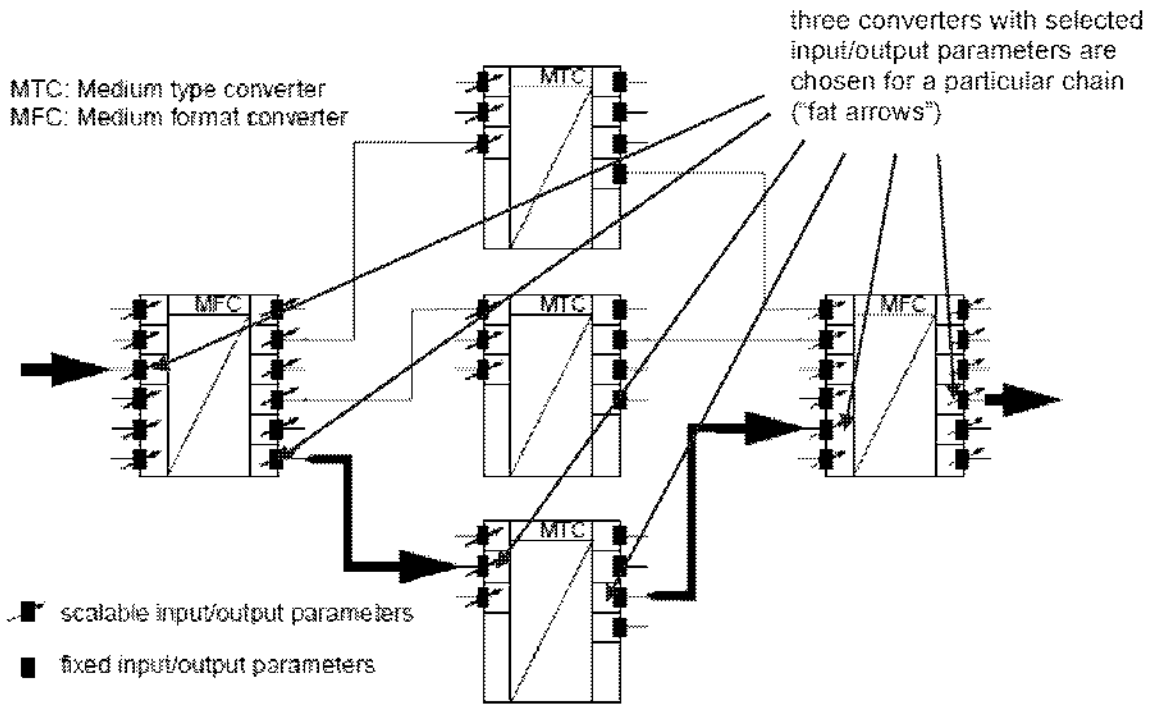
v. “state information”

Claim 1 finally recites: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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 element.

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

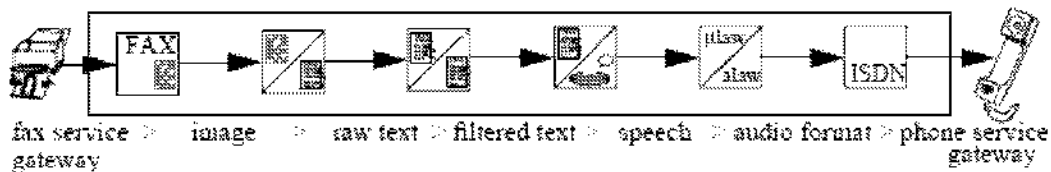
However, Pfeifer96 also discloses this “state information” element under a claim construction that requires component-by-component state information unrelated to the overall sequence of conversion routines.

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. 3* at 108-09.



Id. at 108 (Figure 4: “Medium type conversion with format adaptation”). These parameters control, *e.g.*, “frame/sampling rate, quantization, resolution, size, color depth . . . compression technique” and so on, and they are considered when performing the complex Quality of Service analysis comparing possible candidate chains, described above. *Id.* at 107-08, 114-16; Claim 1(iii) above. Pfeifer96 teaches, for example, that the parameters chosen for the converters across a specific chain should be coordinated to avoid unnecessary loss of information. *See Ex. 3 at 107-08, 115 (e.g., “multiple lossy compression and decompression processes” should be avoided), 124.* These parameters are “state information relating to the processing of the component” which would be used for processing each packet: *e.g.*, to determine which input or output “sampling rate” or “compression” technique should be used. *Id.*

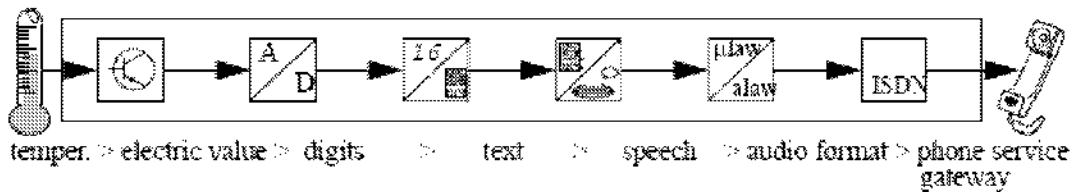
As another example, Pfeifer96 teaches a chain of converters for communicating a fax to a user who has access to a telephone but not a fax machine:



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 components” in this sequence would maintain state information across packets in the manner recited by claim 1, 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 recognize ISDN is a stateful protocol and that this component would therefore maintain state information across packets in order to correctly execute the ISDN protocol.³⁴ As another example, there is a “Text-to-speech” (TTS) component in the chain, and Pfeifer96 teaches that such a component would employ “buffering processes” in order to “wait for the end of a sentence or paragraph before determining the correct prosody” for the generated speech. *Id.* at 115, 125. *See also id.* at 111-12 (“analys[e] the grammatical structure of a sentence” to “improve the prosody of the speech, that is the intonation and phrase melody.”). Thus, at least “a plurality of components” in this sequence would read on this “state information” element.

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

³⁴ *See, e.g.,* Ex. 4 (ISDN98) (“ISDN Primary Rate User-Network Interface Specification”) (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.



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 read on this “state information” element.

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

(b) *Claim 4*

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

Claim 4 recites: “A method in a computer system for processing a message, the message having a plurality of headers, the method comprising . . .” Under Implicit’s apparent claim constructions, Pfeifer96 discloses this element.

As explained above, Pfeifer96 discloses a method “for processing packets of messages,” including voice or fax messages comprised of ISDN packets, multimedia e-mail messages comprised of TCP/IP packets, wireless messages comprised of packets, and multimedia conferencing messages comprised of packets. *See* Claim 1(i) above. Such packets would contain a plurality of headers (*e.g.*, a layer 2 header, a layer 3 header, and so on).

ii. *“dynamically identify a sequence of components”*

Claim 4 further recites: “analyzing the plurality of headers of a first packet of the message to dynamically identify a sequence of components for processing a plurality of packets of the message such that the output format of the components of the sequence match the input

format of the next component in the sequence, wherein analyzing the plurality of headers of the first packet of the message to dynamically identify the sequence of components includes selecting individual components to form the sequence of components after the first packet of the message is received.” Under Implicit’s apparent claim constructions, Pfeifer96 discloses this element.

As explained above, Pfeifer96 discloses a method “for processing packets of messages,” including voice or fax messages comprised of ISDN packets, multimedia e-mail messages comprised of TCP/IP packets, and wireless messages comprised of packets. *See* Claim 1(i) above. Such packets would contain a plurality of headers (*e.g.*, a layer 2 header, a layer 3 header, and so on).

As further explained above, when an incoming connection is initiated (and after the first packet of the message has been received), the iPCSS must determine (1) a data type (*e.g.*, the “medium”) for the incoming message (*e.g.*, is it a voice call, a fax, or an email?), and (2) its intended “recipient” (*i.e.*, the “called party”). *See* Claim 1(iii) above.

One of ordinary skill would recognize that ascertaining the called party would minimally entail analyzing at least a layer 2 and a layer 3 header: *e.g.*, analyzing a layer 3 header to obtain a destination address, and analyzing a layer 2 header to reach the layer 3 header (including, *e.g.*, confirming the validity of a layer 2 frame check sequence).

The other aspects of this claim element are discussed above. *See* Claim 1(iii) above.

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

Claim 4 further recites: “storing an indication of each of the identified components so that the sequence does not need to be re-identified for subsequent packets of the message.” Under Implicit’s apparent claim constructions, Pfeifer96 discloses this element. *See* Claim 1(iv) above.

iv. “state information”

Claim 4 finally recites: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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 element. *See* Claim 1(v) above.

(c) Claim 10

i. “A computer readable storage medium”

Claim 10 recites: “A computer readable storage medium, other than a data transmission medium, containing instructions for processing packets of a message, the instructions comprising at least one computer-executable module configured to” Under Implicit’s apparent claim constructions, Pfeifer96 discloses this element.

Pfeifer96 discloses what is plainly a software-based system, and one of ordinary skill would understand this software would be stored on a “computer readable storage medium, other than a data transmission medium.” *E.g.*, Ex. 3 at 122 (Figure 11), 113 (“converter software”). For example, it would be stored on a hard disk.

Other aspects of this claim element are discussed above. *See* Claim 1(i) above.

ii. Other claim elements

The remaining elements of claim 10 are also disclosed by Pfeifer96. *See* Claim 1 above.

2. Pfeifer96 Renders Obvious Claims 1, 4, and 10 Under § 103

If certain aspects recited in claims 1, 4, and 10 of the ‘857 patent are not deemed to be disclosed or inherent over Pfeifer96, then the inclusion of those aspects certainly would be obvious over Pfeifer96 in light of the background knowledge of one of ordinary skill in the art, under 35 U.S.C. § 103.

(a) Claim 1

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

Claim 1 recites: “A method in a computer system for processing packets of a message, 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. 3 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 recites the method is for processing “packets of a message.”

Pfeifer96 teaches a “universal platform” meant to achieve “universal connectivity” over both “fixed and wireless networks.” *Id.* at 105, 120, 117-18. Thus, as a general matter, it was obvious to support incoming communications from any mainstream device over any mainstream communication medium, including the many 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 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 its support for voice connections using the well-known “ISDN” public telephone network standard,

it was obvious the system should and would be capable of handling such incoming ISDN calls. *Id.* at 109, 111. Since ISDN is inherently packetized,³⁵ such incoming calls would each comprise “packets of a message.”

Relatedly, Pfeifer96 also discloses a Service Gateway for “sending and reception of . . . faxes.” *Id.* at 126. One of ordinary skill would understand that fax machines are also commonly positioned on ISDN networks, so it was obvious the system should and would be capable of handling such fax transmissions. Since ISDN is inherently packetized, such incoming transmissions would each comprise “packets of a message.”

Pfeifer96 also 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 the system should and would support communication with such devices over wireless networks. Since wireless networks are inherently packetized, such incoming communications would comprise “packets of a message.”

As another example, Pfeifer96 discloses a Service Gateway for “reception and delivery of multimedia e-mail.” *Id.* at 126. Since email predominantly transmitted over packet-oriented networks such as TCP/IP, it was obvious the system should and would support such incoming communications, which would each comprise “packets of a message.” *See also id.* at 118 (Figure 9, showing “multimedia e-email” transmitted by a user at a computer).

³⁵ *See, e.g.,* Ex. 4 (ISDN98) (“ISDN Primary Rate User-Network Interface Specification”) (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.

As another example, Pfeifer96 discloses a Service Gateway “for support of multimedia conferencing.” *Id.* at 126. One of ordinary skill would recognize that while it is possible to transmit audio over a non-packetized, analog phone line, the predominant means of transmitting multimedia information is over a packet-oriented network. It was therefore obvious the system should and would support such incoming communications, which would each comprise “packets of a message.”

As a final set of examples, Pfeifer96 discloses various input/output devices which iPCSS would convert from or to. *E.g., id.* at 110 (“Braille output device”), 6 (“temperature” sensor), 118 (“video” camera). Because such devices typically do not have a communications capability of their own, it was obvious they would be connected to computers which would communicate with the outside world through use of a packet-oriented network such TCP/IP. It was therefore obvious the system should and would support such incoming communications, which would each comprise “packets of a message.”

All of the specific instances of connectivity described above are obvious for at least the reason that Pfeifer96 teaches a “universal platform” meant to achieve “universal connectivity” over both “fixed and wireless networks.” *Id.* at 105, 120, 117-18.

ii. “receiving a packet of the message”

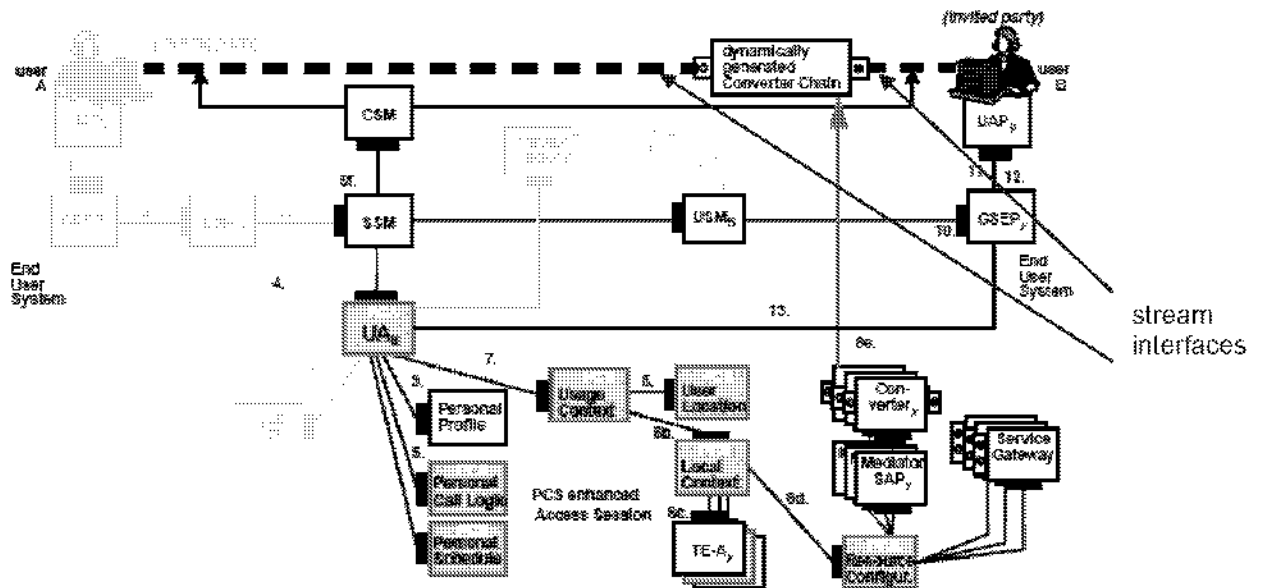
Claim 1 further recites: “receiving a packet of the message and a data type of the message.” Under Implicit’s apparent claim constructions, Pfeifer96 renders obvious this element. *See* Claim 1(i) above and Section V.C.1 (Pfeifer96 102) at Claim 1(ii).

iii. “dynamically identify a sequence of components”

Claim 1 further recites: “analyzing the data type of a first packet of the message to dynamically identify a sequence of components for processing a plurality of packets of the message such that the output format of the components of the sequence match the input format of

the next component in the sequence, wherein analyzing the data type of the first packet of the message to dynamically identify the sequence of components includes selecting individual components to form the sequence of components after the first packet of the message is received.” Under Implicit’s apparent claim constructions, Pfeifer96 renders obvious this element.

As explained above, it was obvious for the system to accept incoming streams of packets from a variety of devices, including computers, ISDN telephones and fax machines, and wireless phones. See Claim 1(i) above. It was therefore also obvious that all of these various types of incoming packet streams should and would be routed to the corresponding dynamically generated converter chain for processing:



Ex. 3 at 122 (Figure 11, showing the “dynamically generated Converter Chain” between the two parties). See also *id.* at 124 (“a dynamically generated converter chain with stream interfaces for the appropriate media is created”).

Other aspects of this claim element are discussed above. See Section V.C.1 (Pfeifer96 102) at Claim 1(iii) above.

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

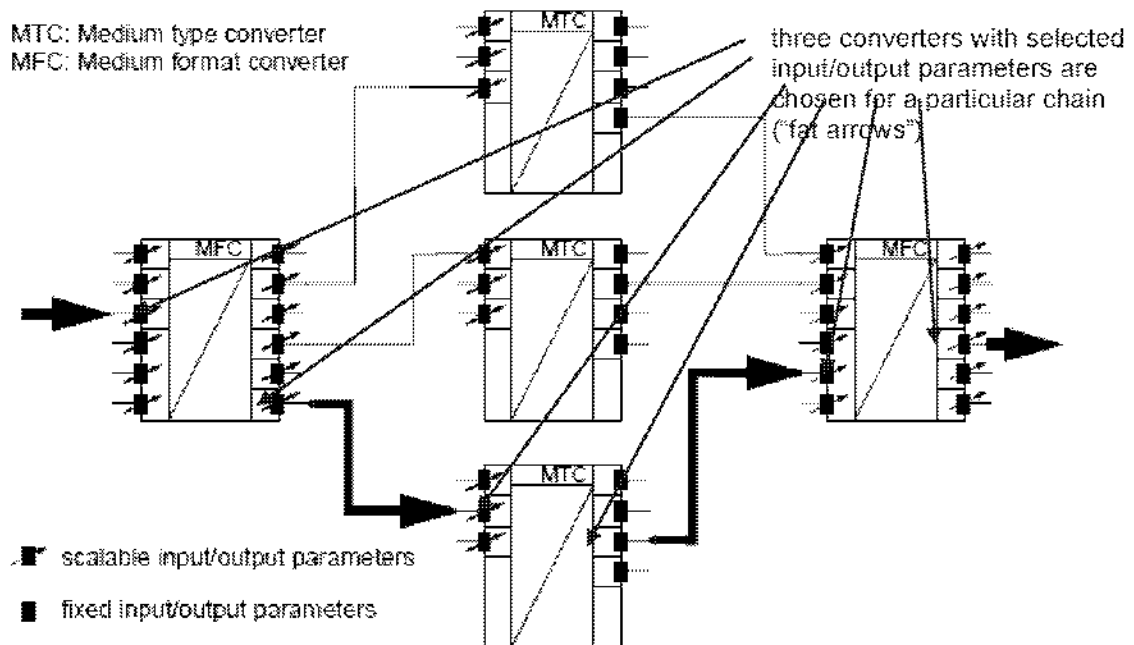
Claim 1 further recites: “storing an indication of each of the identified components so that the sequence does not need to be re-identified for subsequent packets of the message.” Under Implicit’s apparent claim constructions, Pfeifer96 renders obvious this element.

The “dynamically generated converter chain” is generated only once for an incoming communication and is used to process the entire communication. *See* Ex. 3 at 122-27. It was therefore obvious this chain should and 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 under Implicit’s apparent claim constructions. *Id.* at 127.

v. “state information”

Claim 1 finally recites: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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. 3 at 108-09.



Id. at 108 (Figure 4: “Medium type conversion³⁶ with format adaptation”). These parameters control, *e.g.*, “frame/sampling rate, quantization, resolution, size, color depth . . . compression technique” and so on, and they are considered when performing the complex Quality of Service analysis comparing possible candidate chains, described above. *Id.* at 107-08, 114-16; Claim I(iii) above. Pfeifer96 teaches, for example, that the parameters chosen for the converters across a specific chain should be coordinated to avoid unnecessary loss of information. *See* Ex. 3 at 107-08, 115 (*e.g.*, “multiple lossy compression and decompression processes” should be avoided), 124. These parameters are “state information relating to the processing of the component” which would be used for processing each packet: *e.g.*, to determine which input or output “sampling rate” or “compression” technique should be used. *Id.* It was obvious that any or all of the components in a particular chain would maintain such “parameters,” and therefore

³⁶ 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.

obvious that any or all of the components would satisfy the “state information” element of claim 1.

Additionally, 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 expressly disclosed and obvious sequences.

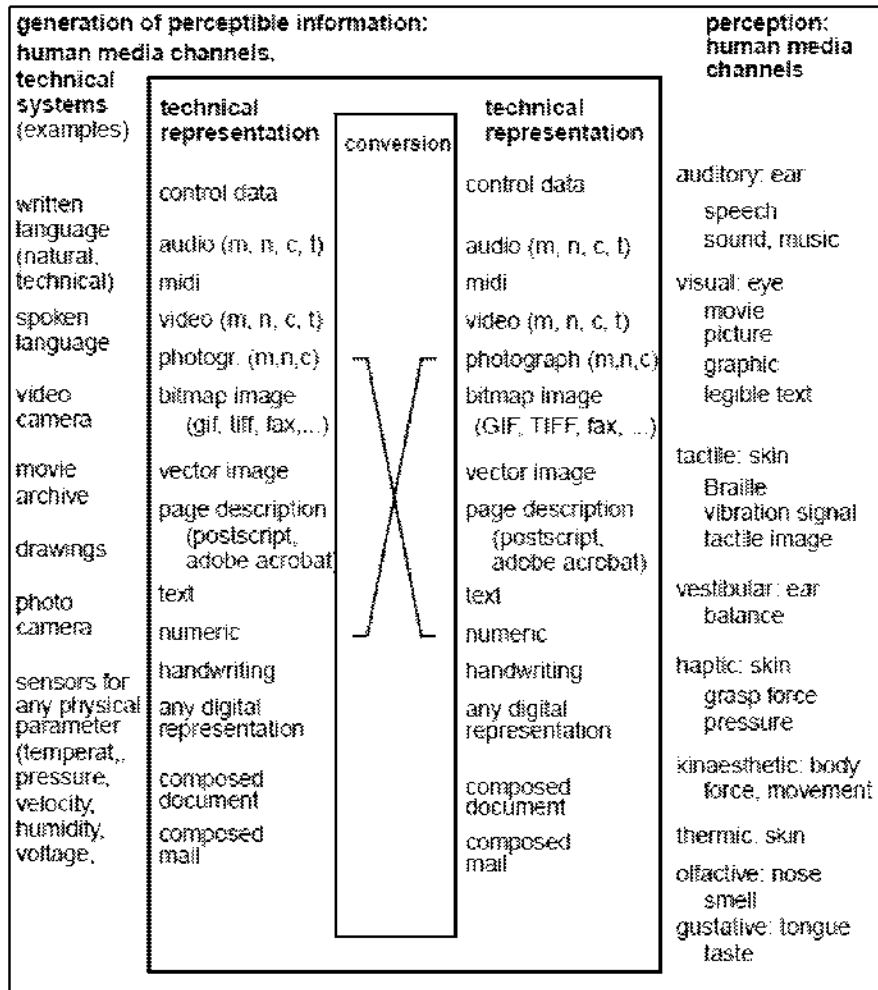
(a) ISDN Adapter Components

Pfeifer96 expressly discloses a component for adapting from audio data to an outgoing ISDN telephone connection, so it was obvious to also employ such an adapter to perform the necessary task of obtaining audio (or other data) from an *incoming* ISDN connection. Ex. 3 at 109 (Figure 5), 111 (Figure 6). Because ISDN is a stateful protocol, it was obvious for such components to maintain state information in the manner recited by claim 1 in order to correctly execute the ISDN protocol.³⁷

(b) Lossless Compression/Decompression Components

Pfeifer96 teaches that its “audio” and “video” converter components would have a settable parameter “c” which specifies the “applied compression technique” employed by the component as part of its conversion:

³⁷ See, e.g., Ex. 4 (ISDN98) (“ISDN Primary Rate User-Network Interface Specification”) (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.



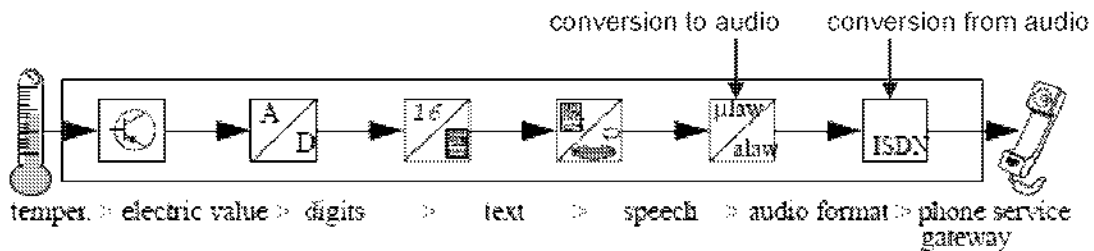
parameters:
m, n: media dependent parameters
(frame/sampling rate, quantization, resolution, size, color depth, etc.)
c: applied compression technique
t: time, duration, etc.

Ex. 3 at 107 (Figure 3: “Generic conversion matrix”). *See also id.* at 108 (Figure 5, showing various “input/output parameters” per converter component) and 115 (“multiple lossy compression and decompression” across a chain should be avoided). Since Pfeifer96 discloses both “input” and “output” parameters and discloses “multiple . . . compression and decompression” operations across a chain, it would be understood (and was obvious) that corresponding decompression techniques would be included in the converter framework. *See id.*

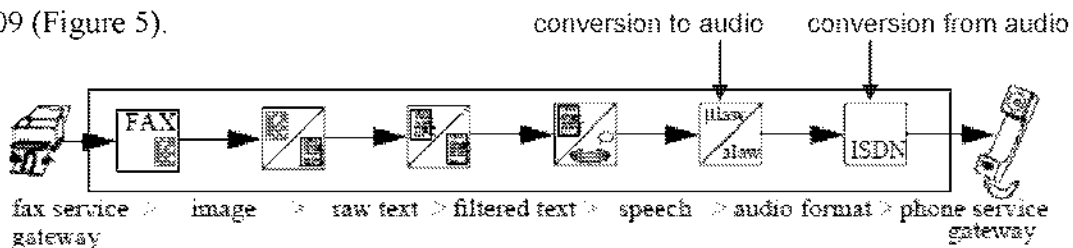
Since Pfeifer96 cautions “lossy” compression/decompression techniques should be avoided but does not specify particular algorithms, one of ordinary skill would draw from

standard background knowledge regarding suitable lossless algorithms that might be applied. *Id.* at 115, 124. In particular, one of ordinary skill would find it obvious to employ an “adaptive” algorithm because they were 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 recited by claim 1.³⁸ Hence, it was obvious that any component converting to or from audio or video would read on the “state information” element of claim 1, by performing adaptive compression or decompression.

Pfeifer96 expressly discloses two converter chains which each contain two components that convert to or from audio, and thus on this basis alone it was obvious that “a plurality of components” in these chains would maintain “state information” across packets in the manner recited by claim 1:



Id. at 109 (Figure 5).



³⁸ See, e.g., Ex. 5 (Nelson) (“The Data Compression Book” by Mark Nelson *et al.*) (1996) at 8 (“Adaptive coding . . . lead[s] to vastly improved compression ratios”), 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 “continually modified as new characters are read in and coded”). This reference is cited in this context solely to help explain Pfeifer96. See MPEP § 2205.

Id. at 111 (Figure 6). Moreover, the final component in each chain is doubly stateful because it would also maintain ISDN connection state across packets, as explained above.

As another example, Pfeifer96 teaches that a user with only a “telephone” may wish to “attend” a “video conference.” *Id.* at 104. *See also, e.g., id.* at 118 (Figure 9: showing “speech” input and “video” output). A simple and 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. Both converters would convert to or from audio, hence it was obvious for every converter component in this chain to maintain “state information” across packets in the manner recited by claim 1. And again, the components in this chain would be doubly stateful because they would also maintain ISDN state across packets, as explained above.

(c) Text-to-Speech Components

Pfeifer96 teaches a “Text-to-speech” (TTS) converter component, and it was obvious for this component to maintain state information across packets in the manner recited by claim 1, because Pfeifer96 expressly discloses “buffering processes” employed by this component to “wait for the end of a sentence or paragraph before determining the correct prosody.” Ex. 3 at 115, 125. *See also id.* at 111-12 (“analys[e] the grammatical structure of a sentence” to “improve the prosody of the speech, that is the intonation and phrase melody.”).

Because a TTS component is included in both of the converter chains depicted in Figures 5 and 6 (discussed above), this is another basis for finding it was obvious that “a plurality of components” in those chains (and in other obvious chains incorporating a TTS component) would read on the “state information” element 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. 3 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. In order for such a component to adaptively improve its recognition of a particular speaker’s voice over time, it was obvious for it to maintain state information across packets in the manner recited by claim 1.

To summarize, it was obvious for many of the converter components disclosed by Pfeifer96 to maintain “state information” across packets in the manner recited by claim 1 in order for them to function as described, including ISDN adapter components, lossless compression/decompression components, TTS components, and speech recognition components. It is also clear that such components comprise “a plurality of components” in certain sequences expressly disclosed by Pfeifer96 (*e.g.*, *id.* at 109, 111), and that they would comprise a “plurality of components” in many other useful and obvious converter sequences.

(b) Claim 4

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

Claim 4 recites: “A method in a computer system for processing a message, the message having a plurality of headers, the method comprising” Under Implicit’s apparent claim constructions, Pfeifer96 renders obvious this element.

As explained above, Pfeifer96 renders obvious a method “for processing packets of messages,” including voice or fax messages comprised of ISDN packets, wireless messages comprised of packets, multimedia e-mail messages comprised of TCP/IP packets, multimedia conferencing messages comprised of packets, and messages from various input/output devices

comprised of packets. *See* Claim 1(i) above. Such packets would contain a plurality of headers (*e.g.*, a layer 2 header, a layer 3 header, and so on).

ii. “dynamically identify a sequence of components”

Claim 4 further recites: “analyzing the plurality of headers of a first packet of the message to dynamically identify a sequence of components for processing a plurality of packets of the message such that the output format of the components of the sequence match the input format of the next component in the sequence, wherein analyzing the plurality of headers of the first packet of the message to dynamically identify the sequence of components includes selecting individual components to form the sequence of components after the first packet of the message is received.” Under Implicit’s apparent claim constructions, Pfeifer96 renders obvious this element.

As explained above, Pfeifer96 discloses a method “for processing packets of messages,” including voice or fax messages comprised of ISDN packets, wireless messages comprised of packets, multimedia e-mail messages comprised of TCP/IP packets, multimedia conferencing messages comprised of packets, and messages from various input/output devices comprised of packets. *See* Claim 1(i) above. Such packets would contain a plurality of headers (*e.g.*, a layer 2 header, a layer 3 header, and so on).

As further explained above, when an incoming connection is initiated (and after the first packet of the message has been received), the iPCSS must determine (1) a data type (*e.g.*, the “medium”) of the incoming message (*e.g.*, is it a voice call, a fax, or an email?), and (2) its intended “recipient” (*i.e.*, the “called party”). *See* Claim 1(iii) above.

One of ordinary skill would find it obvious to ascertain the called party by analyzing a plurality of headers, including at least a layer 2 and a layer 3 header: *e.g.*, analyzing a layer 3 header to obtain a destination address, and analyzing a layer 2 header to reach the layer 3 header

(including, *e.g.*, confirming the validity of a layer 2 frame check sequence). One of ordinary skill would also find it obvious to ascertain a data type by analyzing a plurality of headers: *e.g.*, analyzing a layer 4 header to obtain a destination TCP/IP port, and analyzing the layer 2 and layer 3 headers to reach the layer 4 header.

Other aspects of this claim element are discussed above. *See* Claim 1(iii) above.

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

Claim 4 further recites: “storing an indication of each of the identified components so that the 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. *See* Claim 1(iv) above.

iv. “state information”

Claim 4 finally recites: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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 element. *See* Claim 1(v) above.

(c) Claim 10

i. “A computer readable storage medium”

Claim 10 recites: “A computer readable storage medium, other than a data transmission medium, containing instructions for processing packets of a message, the instructions comprising at least one computer-executable module configured to” Under Implicit’s apparent claim constructions, Pfeifer96 renders obvious this element.

Pfeifer96 discloses what is plainly a software-based system, and one of ordinary skill would understand this software would be stored on a “computer readable storage medium, other

than a data transmission medium.” *E.g.*, Ex. 3 at 122 (Figure 11), 113 (“converter software”). For example, it would be stored on a hard disk.

Other aspects of this claim element are discussed above. *See* Claim 1(i) (showing “processing packets of a message”) above.

ii. Other claim elements

The remaining elements of claim 10 are also rendered obvious by Pfeifer96. *See* Claim 1 above.

3. Pfeifer96 in View of ISDN98 and Nelson Renders Obvious Claims 1, 4, and 10 Under § 103

The specification “ISDN Primary Rate User-Network Interface Specification” (Exhibit 4, “ISDN98”) was published in August 1998 by Northern Telecom. The treatise “The Data Compression Book” (Exhibit 5, “Nelson”) by Mark Nelson *et al.* was published in 1996. Neither was considered during the prosecution of the ‘857 patent.

If certain aspects recited in claims 1, 4, and 10 of the ‘857 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 in light of the background knowledge of one of ordinary skill in the art, under 35 U.S.C. § 103.

It was obvious to supplement the teachings of Pfeifer96 with ISDN98 because Pfeifer96 expressly discloses components for adapting data for an ISDN network interface, and ISDN98 (the “ISDN Primary Rate User-Network Interface Specification”) would have been an obvious place to look for further details regarding the implementation of such ISDN components. Ex. 3 at 109, 111. It was further obvious to supplement these teachings with Nelson, because Pfeifer96 teaches that many of its converters would have a “parameter” specifying “applied compression

technique,” and Nelson (“The Data Compression Book”) would have been an obvious place to look for information regarding specific compression techniques that could be applied. *Id.* at 107.

(a) *Claim 1*

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

Claim 1 recites in pertinent part: “A method in a computer system for processing packets of a message, the method comprising” Under Implicit’s apparent claim constructions, Pfeifer96 in view of ISDN98 and Nelson renders obvious this element.

ISDN98 clarifies that incoming ISDN transmissions (*e.g.*, voice calls or fax transmissions) would comprise “packets of a message.” In a chapter entitled “Layer 2 frame structure,” ISDN98 provides a diagram showing various fields of a layer 2 frame, including a “Frame check sequence.” Ex. 4 at 3-9 to 3-10. Under Implicit’s apparent claim constructions, such discrete frames would comprise “packets.” *See also, e.g., id.* at 4-37 (“Layer 3 message formats”).

ii. “state information”

Claim 1 finally recites in pertinent part: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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 ISDN98 and Nelson renders obvious this element.

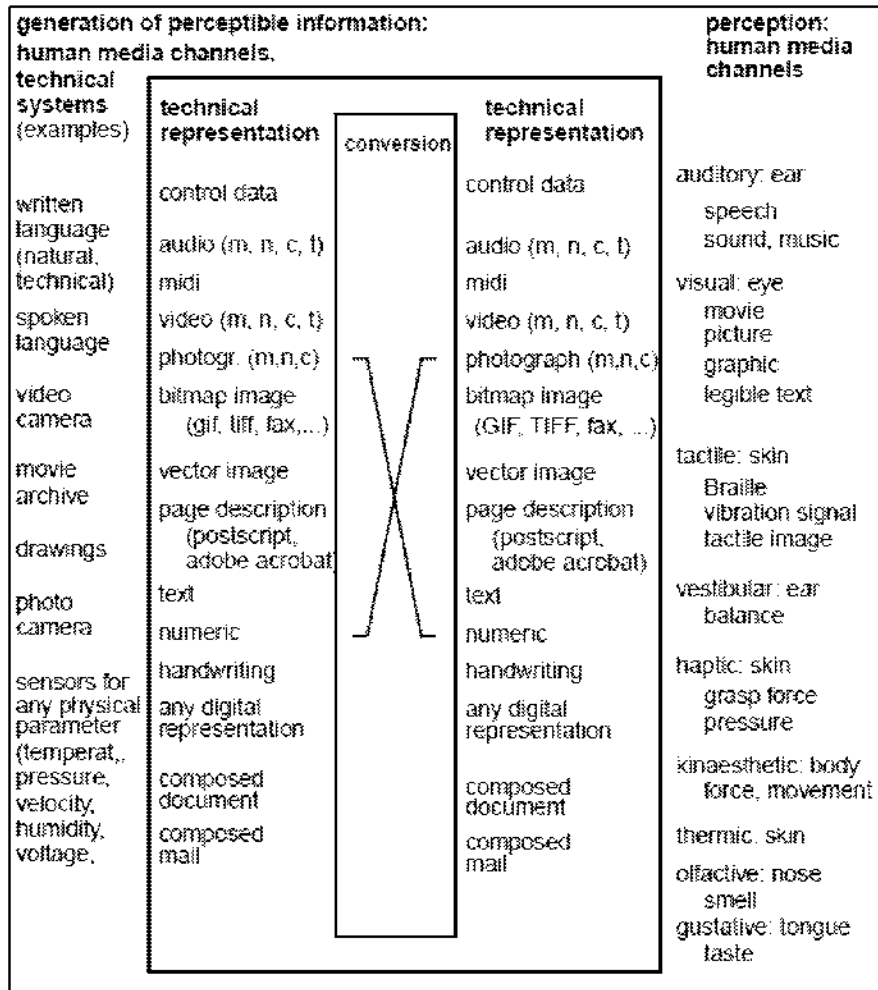
(a) ISDN Adapter Components

Pfeifer96 expressly discloses a component for adapting from audio data to an outgoing ISDN telephone connection, so it was obvious to also employ such an adapter to perform the necessary task of obtaining audio (or other data) from an *incoming* ISDN connection. Ex. 3 at 109 (Figure 5), 111 (Figure 6). ISDN98 confirms ISDN is a stateful protocol, and that it would

be obvious for such components to maintain state information in the manner recited by claim 1 in order to correctly execute the ISDN protocol. For example, ISDN98 teaches that the layer 2 “I format is used for frames that transfer information between Layer 3 entities,” and each I frame has a “send sequence number” and a “receive sequence number.” Ex. 4 at 3-18. Consequently, “[w]hen using I-frame commands, each point-to-point data link connection endpoint” maintains “an associated send state variable” and “an associated acknowledge state variable.” *Id.* at 3-20. *See also, e.g., id.* at 4-54 (“Call state”).

(b) Lossless Compression/Decompression Components

Pfeifer96 teaches that a number of its converter components would have a settable parameter “c” which specifies the “applied compression technique” employed by the component as part of its conversion:



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

Ex. 3 at 107 (Figure 3: “Generic conversion matrix”). *See also id.* at 108 (Figure 5, showing various “input/output parameters” per converter component) and 115 (“multiple lossy compression and decompression” across a chain should be avoided). Since Pfeifer96 discloses both “input” and “output” parameters and discloses “multiple . . . compression and decompression” operations across a chain, it would be understood (and was obvious) that corresponding decompression techniques would be included in the converter framework. *See id.*

Since Pfeifer96 cautions “lossy” compression/decompression techniques should be avoided but does not specify particular algorithms, one of ordinary skill would be motivated to

employ a suitable lossless compression/decompression algorithms for these converter components. *Id.* at 115, 124.

In particular, an “adaptive” algorithm was an obvious choice for such lossless compression/decompression, as taught by Nelson. Nelson explains that “[a]daptive 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), “Dictionary-Based Compression” (chapter 7; *id.* at 203), “Sliding Window Compression” (chapter 8; *id.* at 215); and the prominent “LZ” family of adaptive 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 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).

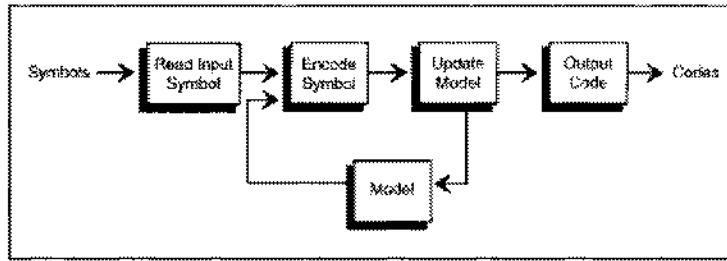


FIGURE 2.2 GENERAL ADAPTIVE COMPRESSION.

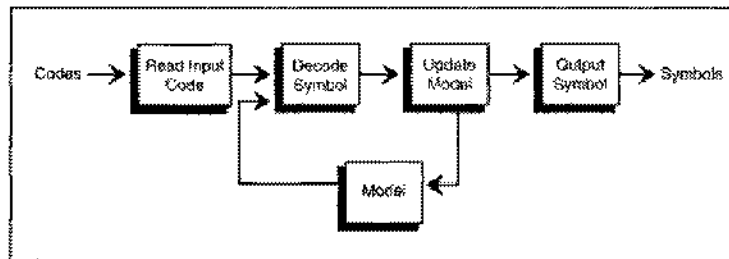
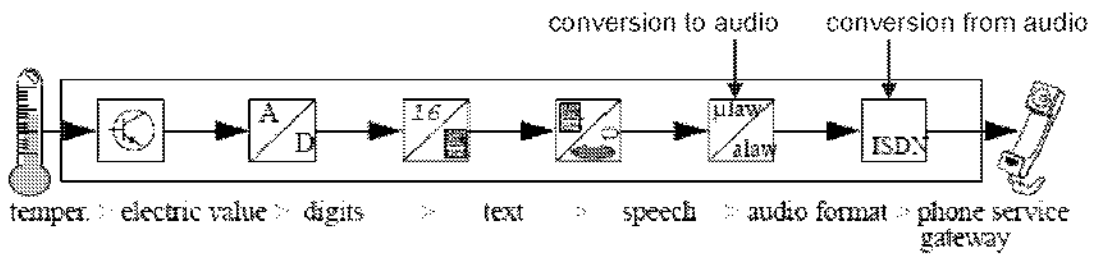


FIGURE 2.3 GENERAL ADAPTIVE DECOMPRESSION.

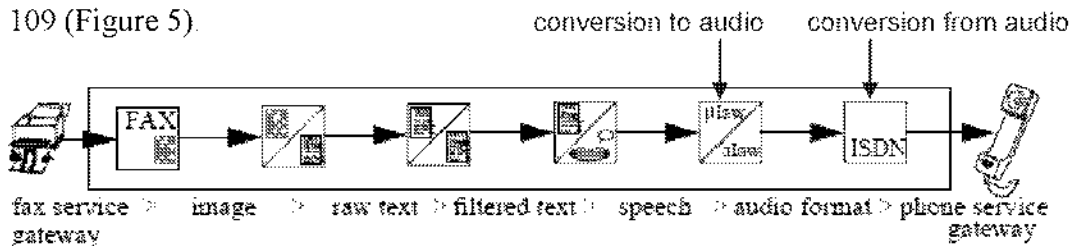
Id. at 19 (showing “Update Model” (state information) after encoding or decoding every piece of data). Nelson explains: “adaptive models start knowing essentially nothing about the data” so “when the program first starts it doesn’t do a very good job of compression.” *Id.* at 19. However, “Most adaptive 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 maintain state information across packets in the manner recited by claim 1, in order to attain these “respectable compression ratios.” *Id.*

Pfeifer96 expressly discloses two converter chains which each contain two components that convert to or from audio, and thus on this basis alone it was obvious that “a plurality of components” in these chains would maintain “state information” across packets in the manner recited by claim 1:



Ex. 3 at 109 (Figure 5).



Id. at 111 (Figure 6). Moreover, the final component in each chain is doubly stateful because it would also maintain ISDN connection state across packets, as explained above.

As another example, Pfeifer96 teaches that a user with only a “telephone” may wish to “attend” a “video conference.” *Id.* at 104. *See also, e.g., id.* at 118 (Figure 9: showing “speech” input and “video” output). A simple and 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. Both converters would convert to or from audio, hence it was obvious for every converter component in this chain to maintain “state information” across packets in the manner recited by claim 1. And again, the components in this chain would be doubly stateful because they would also maintain ISDN state across packets, as explained above.

To summarize, it was obvious for many of the converter components disclosed by Pfeifer96 to maintain “state information” across packets in the manner recited by claim 1 in order for them to function as described, including the ISDN adapter components and lossless compression/decompression components discussed immediately above, and the TTS components and speech recognition components discussed elsewhere above. *See* Section V.C.2 (Pfeifer96) at Claim 1(v) above. It is also clear that such components comprise “a plurality of components” in certain sequences expressly disclosed by Pfeifer96 (*e.g.*, *id.* at 109, 111), and that they would comprise a “plurality of components” in many other useful and obvious converter sequences.

(b) Claim 4

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

Claim 4 recites in pertinent part: “A method in a computer system for processing a message, the message having a plurality of headers, the method comprising” Under Implicit’s apparent claim constructions, Pfeifer96 in view of ISDN98 and Nelson renders obvious this element.

As explained above, these references render obvious a method “for processing packets of messages,” including voice or fax messages comprised of ISDN packets. *See* Claim 1(i) above. Such packets would contain a plurality of headers (*e.g.*, a layer 2 header, a layer 3 header, and so on). *See, e.g.*, Ex. 4 (ISDN98) at 3-9 (“Layer 2 frame structure”), 4-37 (“Layer 3 message formats”).

ii. “state information”

Claim 4 finally recites in pertinent part: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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 ISDN98 and Nelson renders obvious this element. *See* Claim 1(ii) above.

(c) Claim 10

i. “A computer readable storage medium”

Claim 10 recites in pertinent part: “A computer readable storage medium, other than a data transmission medium, containing instructions for processing packets of a message, the instructions comprising at least one computer-executable module configured to” Under Implicit’s apparent claim constructions, Pfeifer96 in view of ISDN98 and Nelson renders obvious this element. *See* Claim 1(i) (showing “processing packets of a message”) above.

ii. “state information”

Claim 10 finally recites in pertinent part: “for each of a plurality of components in the identified sequence: perform the processing of each packet by the identified component; and store 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 ISDN98 and Nelson renders obvious this element. *See* Claim 1(ii) above.

4. Pfeifer96 in View of Arbanowksi96 Renders Obvious Claims 1, 4, and 105 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 ‘857 patent.

If certain aspects recited in claims 1, 4, and 10 of the ‘857 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 in light of the background knowledge of one of ordinary skill in the art, 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 of claims 1, 4, and 10 presented in Section V.C.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 packets of a message, the method comprising . . .” Under Implicit’s apparent claim constructions, Pfeifer96 in view of Arbanowksi96 renders obvious this element.

Arbanowksi96 further confirms a “computer system” would be used. *E.g.*, Ex. 11 at 6-8, 19 (“Computational Objects”).

Claim 1 recites the method is “for processing packets of a message.” 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”). *See also* Ex. 3 (Pfeifer96) at 126 (“Service Gateways” for “phone . . . fax . . . mail . . . multimedia conferencing,” etc.). Thus, an incoming communication over any of those packet-oriented networks would comprise “packets of a message.”

ii. “receiving a packet of the message”

Claim 1 further recites: “receiving a packet of the message and a data type of the message.” Under Implicit’s apparent claim constructions, Pfeifer96 in view of Arbanowski96 renders obvious this element.

As explained in the following section, Arbanowski96 further confirms an incoming data type is crucial for determining the series of conversions needed to deliver the message to its intended recipient at his or her current location. *See also* Claim 1(i) above.

iii. “dynamically identify a sequence of components”

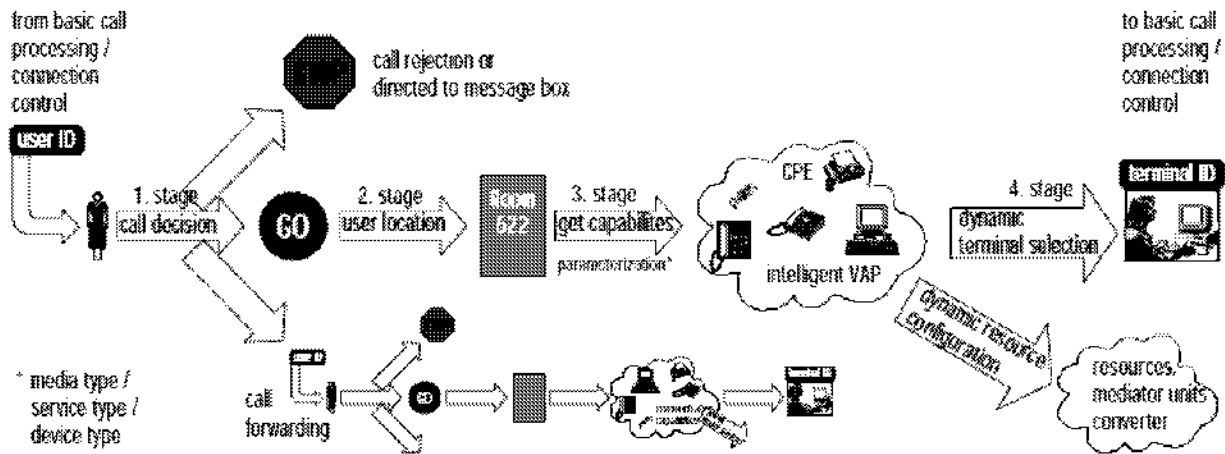
Claim 1 further recites: “analyzing the data type of a first packet of the message to dynamically identify a sequence of components for processing a plurality of packets of the message such that the output format of the components of the sequence match the input format of

the next component in the sequence, wherein analyzing the data type of the first packet of the message to dynamically identify the sequence of components includes selecting individual components to form the sequence of components after the first packet of the message is received.” Under Implicit’s apparent claim constructions, Pfeifer96 in view of Arbanowski96 renders obvious this element.

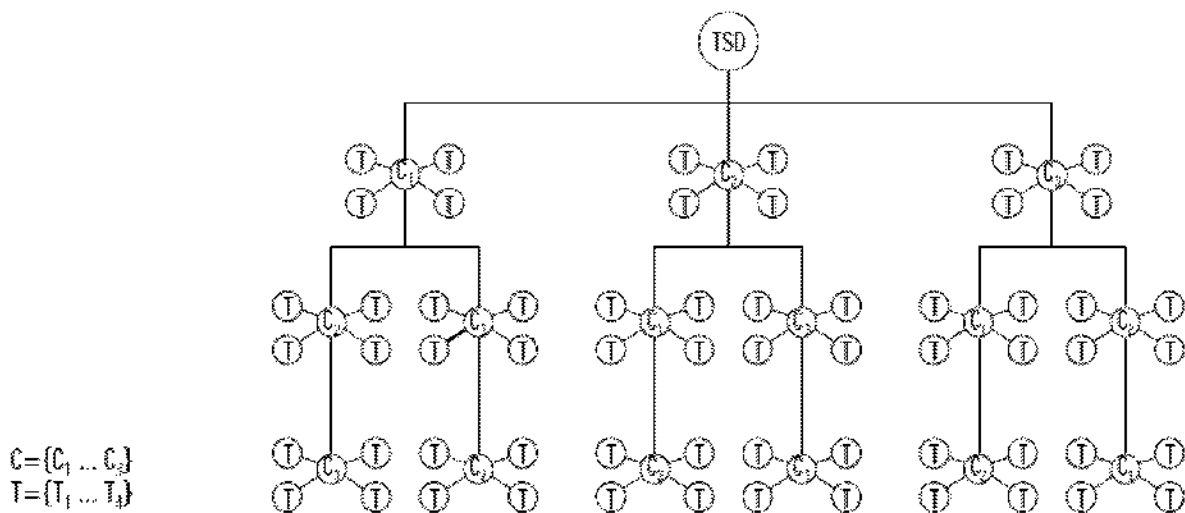
As explained above, when an incoming connection is initiated (*i.e.*, when the first packet of the message has been received by iPCSS), the system must determine (1) a data type (*e.g.*, the “medium”) for the incoming message (*e.g.*, is it a voice call, a fax, or an email?), and (2) its intended “recipient” (*i.e.*, the “called party”). *See* Section V.C.1 (Pfeifer96 102) at Claim 1(iii) above. Arbanowski96 confirms that determining a data type for the incoming message is a crucial part of selecting an appropriate sequence of converters for the message. *E.g.*, Ex. 11 at 13 (“adaptation of the medium type to the available set of terminals”; “In most cases they require conversions of the medium type or at least the medium format, because . . . different services mostly use different media types and media formats.”), 28 (“Medium” being a key attribute of the “Service Description”).

And as suggested by its title (“Generic Description of Telecommunication Services and *Dynamic* Resource Selection in Intelligent Communication Environments”), Arbanowski96 also confirms the sequence of converters is identified *dynamically*.

Like Pfeifer96, Arbanowski96 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,” showing “dynamic resource configuration” in 4th stage). Arbanowski96 provides additional detail on the elaborate process by which many possible candidate chains of converters are assembled and evaluated for their potential “Quality of Service” in the course of connecting the call. *E.g.*, Ex. 11 at 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”), 53 (“All possible converter chains are stored as temporary solutions. The next step is now, to calculate the Quality of Service for every single temporary solution.”).



Id. at 51 (Figure 4-8: “Dynamic Resource Selection – Possible Converter Chains”). Because iPCSS constructs many possible candidate chains and performs a complex Quality of Service analysis on these candidate 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,” under Implicit’s claim constructions. *See id.* at 49-54.

Arbanowski96 also further confirms that iPCSS considers the concatenation of many possible individual converter components 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 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.

As explained by Pfeifer96, once an optimal “converter chain” for the incoming message has been generated, the chain “is instantiated as an object with stream interfaces.” Ex. 3 at 127. Arbanowski96 provides further detail on this process, further confirming this object would endure throughout the connection, *e.g.*: “If a valid converter chain was found, the included converters have to be configured. This means, that streams have to be connected, the quality of service parameters have to be controlled and the connection has to be managed up the end of the session.”). Ex. 11 at 52-53 (at end of section entitled “Calculating the most appropriate chain”).

v. “state information”

Claim 1 finally recites: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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 element.

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.C.2 (Pfeifer96 103) in several ways. *See* Section V.C.2 (Pfeifer96 103) at Claim 1(v).

(b) Claim 4

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

Claim 4 recites: “A method in a computer system for processing a message, the message having a plurality of headers, the method comprising” Under Implicit’s apparent claim constructions, Pfeifer96 in view of Arbanowksi96 renders obvious this element.

As explained above, Pfeifer96 in view of Arbanowski96 renders obvious a method “for processing packets of messages,” including messages arriving via packet-oriented technologies such as “ATM,” “ISDN,” and “Ethernet.” *See* Claim 1(i) above. Such packets would contain a plurality of headers (*e.g.*, a layer 2 header, a layer 3 header, and so on).

ii. “dynamically identify a sequence of components”

Claim 4 further recites: “analyzing the plurality of headers of a first packet of the message to dynamically identify a sequence of components for processing a plurality of packets

of the message such that the output format of the components of the sequence match the input format of the next component in the sequence, wherein analyzing the plurality of headers of the first packet of the message to dynamically identify the sequence of components includes selecting individual components to form the sequence of components after the first packet of the message is received.” Under Implicit’s apparent claim constructions, Pfeifer96 in view of Arbanowski96 renders obvious this element.

As explained above, when an incoming connection is initiated (and after the first packet of the message has been received), the iPCSS must determine (1) a data type (*e.g.*, the “medium”) of the incoming message (*e.g.*, is it a voice call, a fax, or an email?); and (2) its intended “recipient” (*i.e.*, the “called party”). *See* Section V.C.1 (Pfeifer96 102) at Claim 1(iii) above. Arbanowski96 confirms that determining the identity of the called party is a crucial part of selecting an appropriate sequence of converters for the message; once determined it is internally represented by a “Personal Identifier (PID),” and this PID drives the selection of suitable terminal for receiving the message in the called party’s current location. Ex. 11 at 1 (“PID”), 6 (“multi stage functional mapping from the PID of the called party . . . to a physical terminal . . . at the location of the called person”).

One of ordinary skill would find it obvious to ascertain the called party by analyzing a plurality of headers, including at least a layer 2 and a layer 3 header: *e.g.*, analyzing a layer 3 header to obtain a destination address, and analyzing a layer 2 header to reach the layer 3 header (including, *e.g.*, confirming the validity of a layer 2 frame check sequence). One of ordinary skill would also find it obvious to ascertain a data type by analyzing a plurality of headers: *e.g.*, analyzing a layer 4 header to obtain a destination TCP/IP port, and analyzing the layer 2 and layer 3 headers to reach the layer 4 header.

Other aspects of this claim element are discussed above. *See* Claim 1(iii) above.

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

Claim 4 further recites: “storing an indication of each of the identified components so that the sequence does not need to be re-identified for subsequent packets of the message.” Under Implicit’s apparent claim constructions, Pfeifer96 in view of Arbanowksi96 renders obvious this element. *See* Claim 1(iv) above.

iv. “state information”

Claim 4 finally recites: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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 element. *See* Claim 1(v) above.

(c) Claim 10

i. “A computer readable storage medium”

Claim 10 recites: “A computer readable storage medium, other than a data transmission medium, containing instructions for processing packets of a message, the instructions comprising at least one computer-executable module configured to” Under Implicit’s apparent claim constructions, Pfeifer96 in view of Arbanowski96 renders obvious this element. *See* Claim 1(i) (showing “processing packets of a message”) above.

ii. Other claim elements

The remaining elements of claim 10 are also rendered obvious by Pfeifer96 in view of Arbanowski96. *See* Claim 1 above.

5. Pfeifer96 in View of Pfeifer97 Renders Obvious Claims 1, 4, and 10 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 ‘857 patent.

If certain aspects recited in claims 1, 4, and 10 of the ‘857 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 in light of the background knowledge of one of ordinary skill in the art, 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 of claims 1, 4, and 10 presented in Section V.C.2 (Pfeifer96 103) in many ways. Some exemplary aspects of Pfeifer97 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 packets of a message, the method comprising . . .” Under Implicit’s apparent claim constructions, Pfeifer96 in view of Pfeifer97 renders obvious this element.

Pfeifer97 further confirms a “computer system” would be used. *I.g.*, Ex. 12 at 150-52 (“Computational Model”).

Claim 1 recites the method is “for processing packets of a message.” 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 138 (Table 6: “Possible Values for the Attribute Bearer”). *See also id.* at 151 (Figure 9, showing “Service Gateways”); Ex. 3 (Pfeifer96) at 126 (“Service Gateways” for “phone . . . fax . . . mail . . . multimedia conferencing,” etc.). Thus, an incoming communication over any of those packet-oriented networks would comprise “a message having a sequence of packets.”

ii. “receiving a packet of the message”

Claim 1 further recites: “receiving a packet of the message and a data type of the message.” Under Implicit’s apparent claim constructions, Pfeifer96 in view of Pfeifer97 renders obvious this element.

As explained in the following section, Pfeifer97 further confirms an incoming data type is crucial for determining the series of conversions needed to deliver the message to its intended recipient at his or her current location. *See also* Claim 1(i) above.

iii. “dynamically identify a sequence of components”

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

As explained above, when an incoming connection is initiated (*i.e.*, when the first packet of the message has been received by iPCSS), the system must determine (1) a data type (*e.g.*, the “medium”) for the incoming message (*e.g.*, is it a voice call, a fax, or an email?); and (2) its intended “recipient” (*i.e.*, the “called party”). *See* Section V.C.1 (Pfeifer96 102) at Claim 1(iii) above.

Pfeifer97 confirms that determining a data type for the incoming message is a crucial part of selecting an appropriate sequence of converters for the message. *E.g.*, Ex. 12 at 133 (“automated, intelligent decisions how to . . . bridge any form of communication with any other For automated handling, the heterogeneity of communication has to be classified precisely, allowing the system to match components for various purposes within a huge construction kit”), 134-35 (“Medium” being a key attribute of the “Teleservice Descriptor”).

Pfeifer97 also confirms the sequence of converters is identified *dynamically*. Pfeifer97 teaches that as part of the “Automatic Resource Selection” performed by iPCSS, the system will “find the most appropriate terminal for a requested Teleservice at the [called] user’s current location dynamically.” *Id.* at 143. Pfeifer97 provides additional detail on the elaborate process

by which many possible candidate chains of converters to a terminal device are assembled and evaluated for their potential “Quality of Service” in the course of connecting the call. *I.g., id.* at 147 (“Calculating the Most Appropriate Device”), 148 (“Finding a Possible Converter Chain”: “Each possible combination of converters with a final terminal [device] must be evaluated”; “three converters and four terminals” lead to “60 possible converter chains”), 149 (“Calculating the most appropriate chain”: “The temporary solutions are store and analyzed the next step is to calculate the QoS [Quality of Service] for every single temporary solution”). Because iPCSS constructs many possible candidate chains and performs a complex Quality of Service analysis on these candidate 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,” under Implicit’s claim constructions. *See id.* at 146-50.

Pfeifer97 also confirms the converter components are arranged “such that the output format of the components of the sequence match the input format of the next component in the sequence.” *Id.* at 146 (“***The converters connected in series have only to meet one condition: The Teleservice produced by the output of a connected converter has to match the Teleservice of the input of its successor.***”).

Pfeifer97 also confirms that iPCSS considers the concatenation of many possible individual converter components in the course of connecting a call, and that the ultimately selected chain will have been composed by selecting individual components. *I.g., id.* at 148 (“Finding a Possible Converter Chain”: “three converters and four terminals” lead to “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 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.

As explained by Pfeifer96, once an optimal "converter chain" for the incoming message has been generated, the chain "is instantiated as an object with stream interfaces." Ex. 3 at 127. Pfeifer97 provides further detail on this process, further confirming this object would endure throughout the connection, *e.g.*: "If a valid converter chain was found, the included converters have to be configured. This means, that streams have to be connected, the QoS [Quality of Service] parameters have to be controlled and the connection has to be managed up the end of the session." Ex. 12 at 149-50 (at end of section entitled "Calculating the most appropriate chain"). Thus, an indication of the identified components is stored so it can be used for the entirety of the session.

v. "state information"

Claim 1 finally recites: "for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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 element.

Pfeifer97 provides additional detail on the types of conversions and converter chains that might be assembled (*e.g.*, Ex. 12 at 135-36, 142-43), 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.C.2 (Pfeifer96 103) in several ways. *See* Section V.C.2 (Pfeifer96 103) at Claim 1(v).

(b) Claim 4

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

Claim 4 recites: “A method in a computer system for processing a message, the message having a plurality of headers, the method comprising . . .” Under Implicit’s apparent claim constructions, Pfeifer96 in view of Pfeifer97 renders obvious this element.

As explained above, Pfeifer96 in view of Pfeifer97 renders obvious a method “for processing packets of messages,” including messages arriving via packet-oriented technologies such as “ATM,” “ISDN,” and “Ethernet.” See Claim 1(i) above. Such packets would contain a plurality of headers (*e.g.*, a layer 2 header, a layer 3 header, and so on).

ii. “dynamically identify a sequence of components”

Claim 4 further recites: “analyzing the plurality of headers of a first packet of the message to dynamically identify a sequence of components for processing a plurality of packets of the message such that the output format of the components of the sequence match the input format of the next component in the sequence, wherein analyzing the plurality of headers of the first packet of the message to dynamically identify the sequence of components includes selecting individual components to form the sequence of components after the first packet of the message is received.” Under Implicit’s apparent claim constructions, Pfeifer96 in view of Pfeifer97 renders obvious this element.

As explained above, when an incoming connection is initiated (and after the first packet of the message has been received), the iPCSS must determine (1) a data type (*e.g.*, the “medium”) of the incoming message (*e.g.*, is it a voice call, a fax, or an email?); and (2) its intended “recipient” (*i.e.*, the “called party”). See Section V.C.1 (Pfeifer96 102) at Claim 1(iii) above. Pfeifer97 confirms that determining the identity of the called party is a crucial part of selecting an appropriate sequence of converters for the message. *E.g.*, Ex. 12 at 143 (“find the

most appropriate terminal for a requested Teleservice at the [called] user's current location dynamically”).

One of ordinary skill would find it obvious to ascertain the called party by analyzing a plurality of headers, including at least a layer 2 and a layer 3 header: *e.g.*, analyzing a layer 3 header to obtain a destination address, and analyzing a layer 2 header to reach the layer 3 header (including, *e.g.*, confirming the validity of a layer 2 frame check sequence). One of ordinary skill would also find it obvious to ascertain a data type by analyzing a plurality of headers: *e.g.*, analyzing a layer 4 header to obtain a destination TCP/IP port, and analyzing the layer 2 and layer 3 headers to reach the layer 4 header.

Other aspects of this claim element are discussed above. *See* Claim 1(iii) above.

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

Claim 4 further recites: “storing an indication of each of the identified components so that the 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. *See* Claim 1(iv) above.

iv. “state information”

Claim 4 finally recites: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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 element. *See* Claim 1(v) above.

(c) Claim 10

i. "A computer readable storage medium"

Claim 10 recites: "A computer readable storage medium, other than a data transmission medium, containing instructions for processing packets of a message, the instructions comprising at least one computer-executable module configured to" Under Implicit's apparent claim constructions, Pfeifer96 in view of Pfeifer97 renders obvious this element. *See* Claim 1(i) (showing "processing packets of a message") above.

ii. Other claim elements

The remaining elements of claim 10 are also rendered obvious by Pfeifer96 in view of Pfeifer97. *See* Claim 1 above.

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 6, "Cox") was published on June 28, 1996. It was not considered during prosecution of the '857 patent.

If certain aspects recited in claims 1, 4, and 10 of the '857 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 in light of the background knowledge of one of ordinary skill in the art, under 35 U.S.C. § 103.

(a) Claim 1

Claim 1 recites in pertinent part: "for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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 general 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. 6 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. *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.” *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:

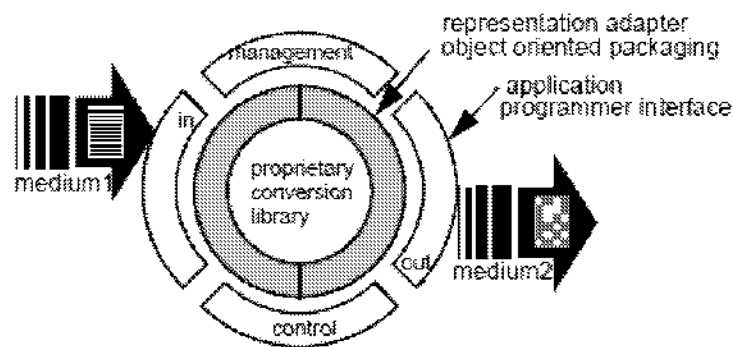
Instead of paying a large fee up-front, all customers, large and small, get the component for free. Later, when they begin to sell their own products based on this component, they pay a (negotiated) fee for using their subvendor's product. The subvendor now receives a continuing revenue stream that is 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. *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”). The excerpts cited above capture only a small portion of the extensive case made by the book for this solution.

One of ordinary skill in the art would readily see the relevance of Cox 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 obvious candidates for this treatment, because Pfeifer96 expressly teaches they may be “proprietary” external components obtained “from different manufacturers,” rather than components developed internally. *Ex. 3 (Pfeifer96)* at 108, 113-14.



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

These converters of Pfeifer96 thus epitomize the “small-granularity reusable software components” discussed throughout Cox. *E.g., Ex. 6* 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. 3 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. 6 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.* Without such an approach, potential vendors of such proprietary conversion libraries would experience precisely the economic difficulties facing Vendor E in the example above, and thus few such libraries would be available to iPCSS. *See id.* at 153-55.

Every distinct product being billed with the Cox technique requires its own invocation count, and it was obvious for any or all of the converters in a particular chain to employ the technique. *E.g.*, Ex. 6 (Cox) at 174-76. Each converter could come from a different vendor (who must be separately paid), and even converters from the same vendor may be priced differently (*e.g.*, depending on complexity). *E.g.*, Ex. 3 (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)”).

Maintaining its invocation count would be an integral part of the processing performed by each component, since without it the economic model fails and iPCSS would lose the use of the component. And maintaining an invocation count per component would clearly entail (as recited by claim 1) “performing the processing of the each packet by the identified component; and storing state information relating to the processing of the component with the packet for use

when processing the next packet of the message”—when the count will be retrieved and incremented again.

(b) Claim 4

Claim 4 recites in pertinent part: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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. *See* Claim 1 above.

(c) Claim 10

Claim 10 recites in pertinent part: “for each of a plurality of components in the identified sequence: perform the processing of each packet by the identified component; and store 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.

7. Pfeifer96 in View of Meer96 Renders Obvious Claims 1, 4, and 10 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, 4, and 10 of the ‘857 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 in light of the background knowledge of one of ordinary skill in the art, under 35 U.S.C. § 103.

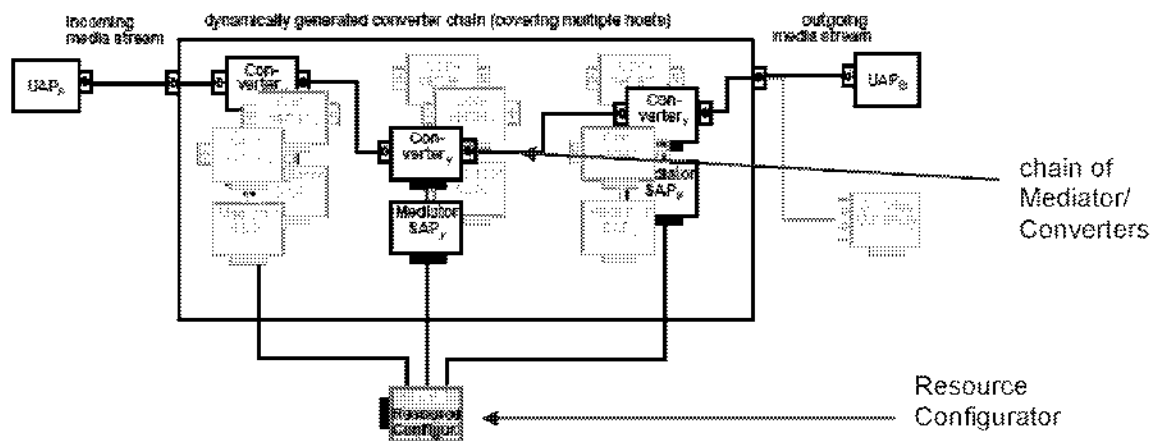
It was obvious to consider Pfeifer96 in view of Meer96 because both documents describe the “Intelligent Personal Communication Support System (iPCSS),” and Meer96 cites to Pfeifer96 for certain concepts. See Ex. 8 at 15 (citations to “Pfeifer96b”), 123 (“Pfeifer96b”: “Generic Conversion of Communication Media for supporting Personal Mobility. To appear in the proceedings of the Third COST 237 Workshop . . . Nov 25-27, 1996”).

Because Pfeifer96 and Meer96 are so similar in approach and detail, Meer96 reinforces and confirms the analysis of claims 1, 4, and 10 presented in Section V.C.2 (Pfeifer96 103) in many ways. *E.g.*, Ex. 8 (Meer96) at 10 (“Dynamic Terminal Selection”), 14 (“Theory of Conversion”), 69 (“Resource Configurator”). Some particularly pertinent aspects of Meer96 are pointed out below.

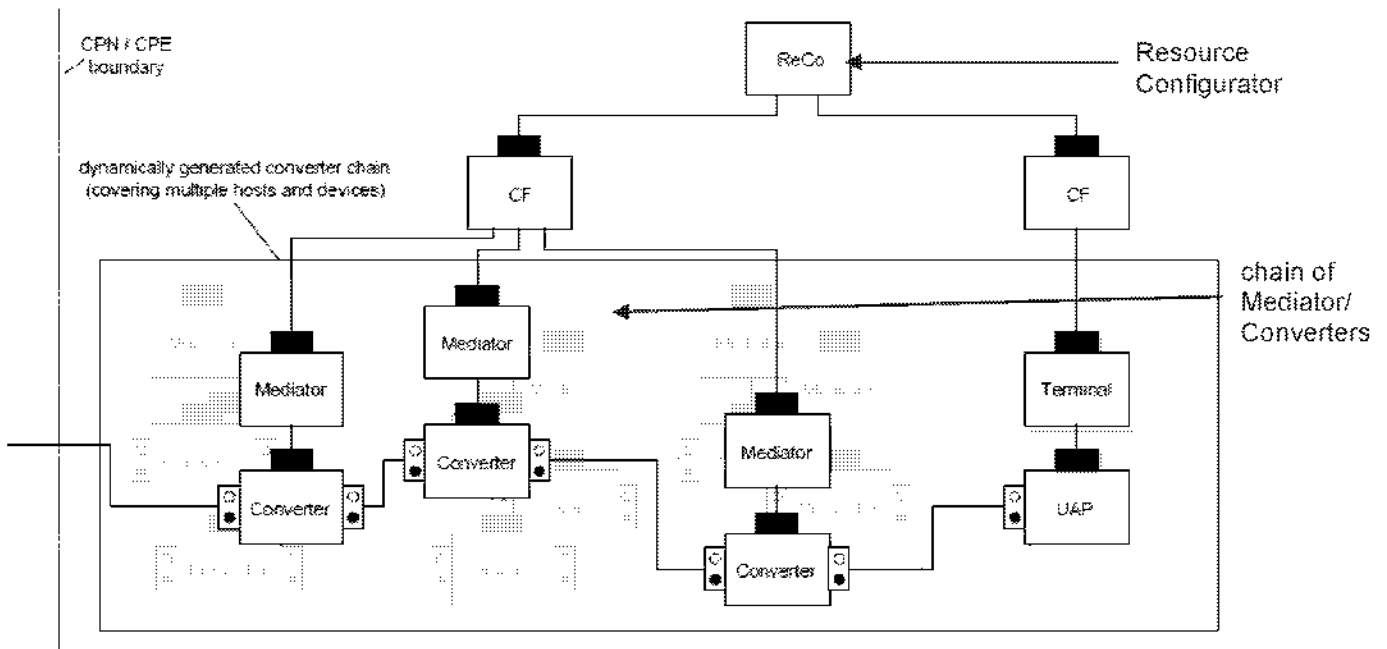
(a) Claim 1

Claim 1 recites in pertinent part: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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. 3 (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 id.* at 124 (“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,” while each “Mediator” is “controlling software” which can “parameterize” its Converter and “configure [its] stream interfaces”).

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 (Figure 4-17), 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 “[n]etwork 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 “[f]uture operating systems.” *Id.* at 54-55.

Thus, using one of these 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 *Id.* at 54. 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 (unlike UDP) it provides for “reliable data transmission with sophisticated error detection and correction,” 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.* at 54, 112.

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 the

desirability of maintaining an *ongoing TCP connection* to the converter, for reasons of performance. 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 clearly 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 entail maintaining “state information” across packets in the manner recited by claim 1. *See, e.g., id.* at 53, 73-74. For example, TCP includes an outgoing sequence number with each packet,³⁹ so transmitting a packet of the message from a mediator to a remote converter module would entail retrieving the most recent sequence number, advancing it, and storing the updated result so it could be used when processing the next packet of the message. The sequence number is “state information relating to the processing of the component” because it is used to transmit message data to the remote converter module for processing.

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

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.

³⁹ *See, e.g.,* Ex. 9 (RFC 793) (“Transmission Control Protocol” [TCP] Specification) (1981) 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.

(b) Claim 4

Claim 4 recites in pertinent part: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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 Meer96 renders obvious this element. *See* Claim 1 above.

(c) Claim 10

Claim 10 recites in pertinent part: “for each of a plurality of components in the identified sequence: perform the processing of each packet by the identified component; and store 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 Meer96 renders obvious this element. *See* Claim 1 above.

8. Pfeifer96 in View of Meer96 and RFC 793 Renders Obvious Claims 1, 4, and 10 Under § 103

The specification RFC 793: “Transmission Control Protocol” (Exhibit 9, “RFC 793”) by the Information Sciences Institute was published in September 1981. It was not considered during prosecution of the ‘857 patent.

If certain aspects recited in claims 1, 4, and 10 of the ‘857 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 in light of the background knowledge of one of ordinary skill in the art, under 35 U.S.C. § 103.

RFC 793 was cited immediately above under MPEP § 2205 as confirming certain background knowledge regarding the stateful operation of TCP. *See* Section V.C.7 (Pfeifer96+Meer96) above.

It was also obvious to consider Pfeifer96 and Meer96 in view of RFC 793. As explained above, TCP was an obvious choice for communicating with remote converter modules using any of the three methods taught by Meer96. *See* Section V.C.7 (Pfeifer96+Meer96) above. Since RFC 793 is the TCP specification, it would have been an obvious place to look for information regarding the specific operation of that protocol.

In particular, RFC 793 confirms that maintaining an ongoing TCP connection to a remote converter module would entail maintaining state across packets in the manner recited by claims 1, 4, and 10. *See* Ex. 9 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.”); Section V.C.7 (Pfeifer96+Meer96) above.

9. Pfeifer96 in View of Franz98 Renders Obvious Claims 1, 4, and 10 Under § 103

The dissertation “Job and Stream Control in Heterogeneous Hardware and Software Architectures” by Stefan Franz (Exhibit 7, “Franz98”) was published on April 22, 1998, and it was not considered during prosecution of the ‘857 patent.

If certain aspects recited in claims 1, 4, and 10 of the ‘857 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 in light of the background knowledge of one of ordinary skill in the art, under 35 U.S.C. § 103.

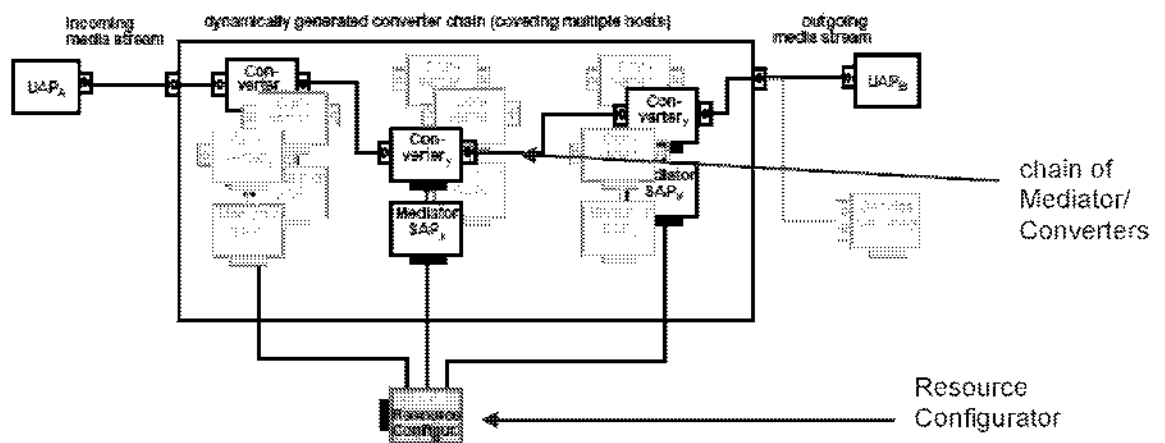
It was obvious to apply Franz98 to Pfeifer96 because the documents both describe the “Intelligent Personal Communication Support System (iPCSS),” with Franz98 presenting further detail regarding the control of jobs and streams in the iPCSS architecture. *See, e.g.*, Ex. 7 at v, 91-94.

Because Pfeifer96 and Franz98 are so similar in approach and detail, Franz98 reinforces and confirms the analysis of claims 1, 4, and 10 presented in Section V.C.2 (Pfeifer96 103) in many ways. *E.g.*, Ex. 7 (Franz98) at 7-16 (including “Resource Configurator” and “Converter Framework”), 91-94 (including “Generic Converter Interface”). Some particularly pertinent aspects of Franz98 are pointed out below.

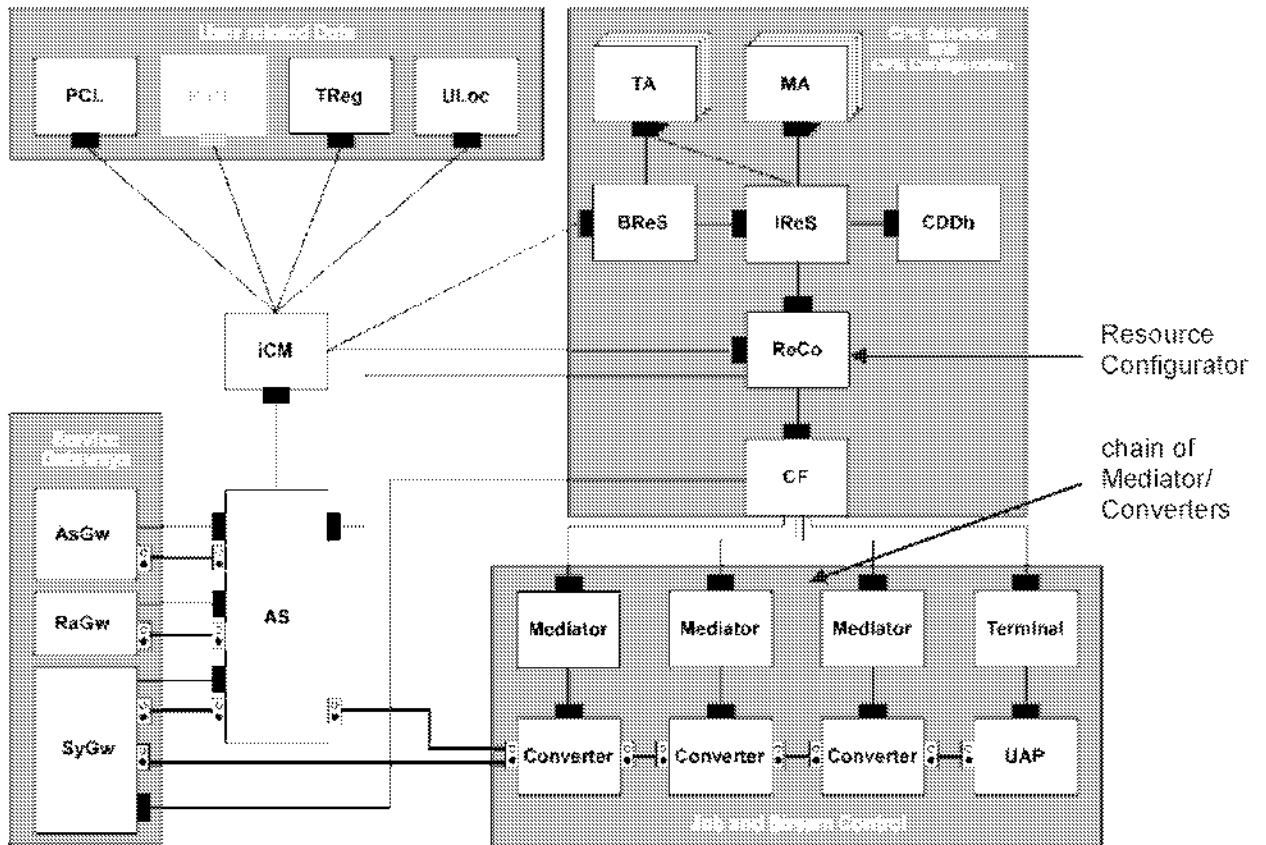
(a) Claim 1

Claim 1 recites in pertinent part: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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. 3 (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 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 id.* at 101-06.

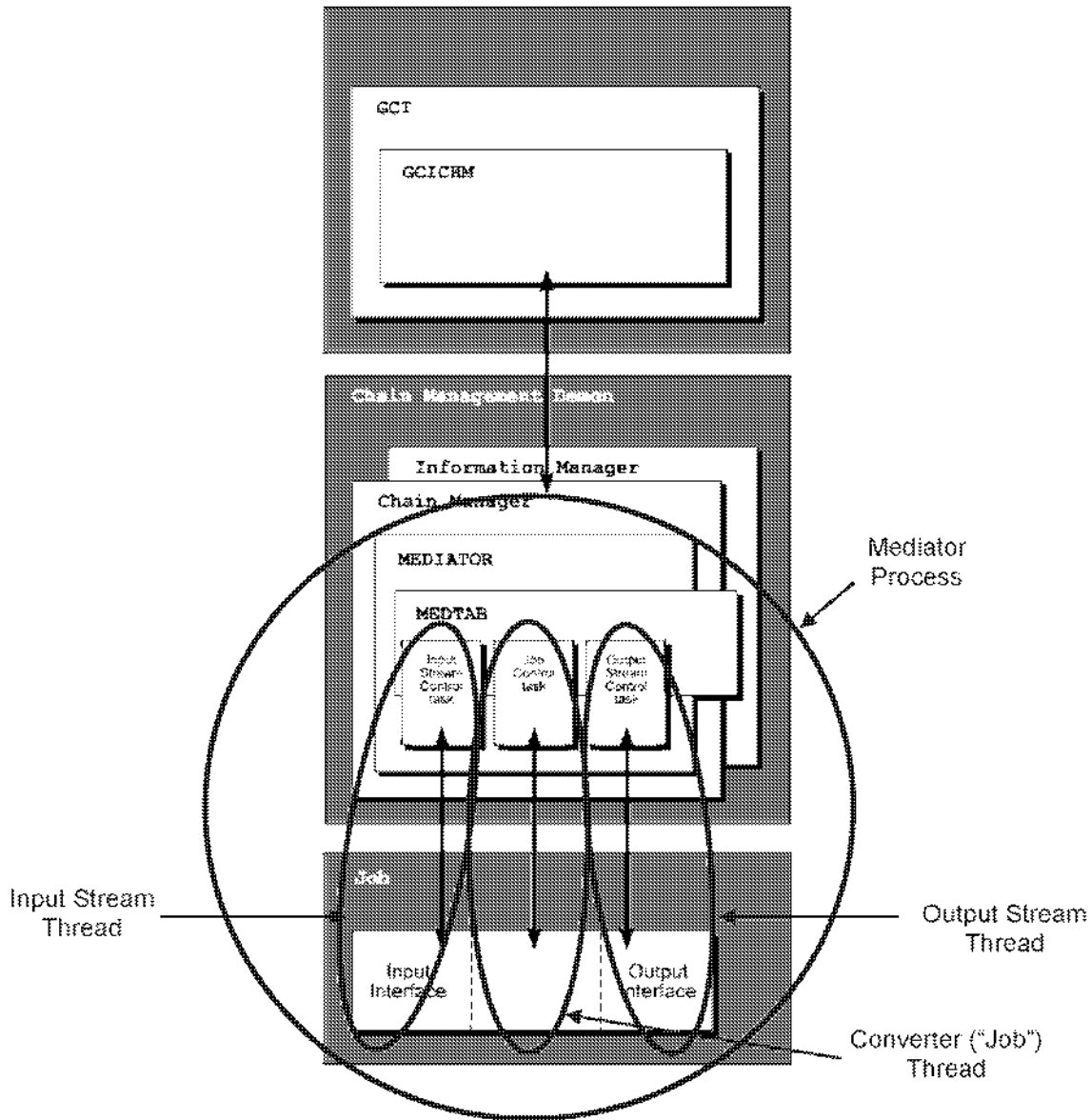
Franz98 presents an elaborate, systematic analysis of the various software building blocks that would be needed for “Job Control and Stream Control” in the iPCSS. *E.g., id.* at 50 (“Programs and Jobs” section), 51 (“Processes” section), 55 (“Threads” section), 58 (“The File System” section).

At the end of this lengthy analysis, Franz98 presents its conclusions regarding how 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. And regarding the “three threads”: “one of these threads is associated to the Job Control [converter] whereas the other two implement the Stream Control” for the converter’s “input stream and . . . output stream.” *Id.*



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 the 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 in original). 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 restored when execution of the thread resumes). *Id.* at 56 (emphasis added).

Thus, 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”) is stored and subsequently re-loaded 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 the processing of the component”; indeed, the program counter and registers would change in response to virtually 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 4

Claim 4 recites in pertinent part: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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 Franz98 renders obvious this element. *See* Claim 1 above.

(c) Claim 10

Claim 10 recites in pertinent part: “for each of a plurality of components in the identified sequence: perform the processing of each packet by the identified component; and store 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 Franz98 renders obvious this element. *See* Claim 1 above.

10. Pfeifer96 in View of ISDN98, Nelson, Cox, Meer96, RFC 793, and Franz98 Renders Obvious Claims 1, 4, and 10 Under § 103

If certain aspects recited in claims 1, 4, and 10 of the ‘857 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.

Claims 1, 4, and 10 recite “dynamically identify a sequence of components.”

Pfeifer96 teaches an “iPCSS” system which “dynamically generate[s]” a chain of converter components after the first packet of a message is received. Ex. 3 at 124.

Claims 1, 4, and 10 recite elements regarding “state information” as relating to “a plurality of components.”

Pfeifer96 teaches converter components that would maintain “state information” across packets in the manner recited by claims 1, 4, and 10: *e.g.*, components for adapting ISDN connections, and components which perform compression/decompression. **ISDN98** confirms the obviousness of employing stateful ISDN protocol adapter algorithms which would read on these elements. **Nelson** confirms the obviousness of employing stateful compression/decompression algorithms which would read on these elements.

Cox teaches an “invocation-based metering” technique which was obvious to apply to the converter components of Pfeifer96, and this technique would read on these elements.

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). **RFC 793** confirms the obviousness of employing a stateful connection algorithm in this context, which would read on these elements.

Franz98 explains that in the iPCSS system, each converter component would maintain state information across packets in a manner which would read on these elements, because of the operating system “threading” structure used for the converter component jobs.

In short, there is no aspect of claims 1, 4, and 10 which was not obvious over the prior art and combinations cited herein.

11. Pfeifer96 in View of Wetherall Renders Obvious Claims 1, 4, and 10 Under § 103

The article “The Active IP Option” (Exhibit 47, “Wetherall”) by David J. Wetherall and David L. Tennenhouse was published by September 11, 1996. It was not considered during the prosecution of the ‘857 patent.

If certain aspects recited in claims 1, 4, and 10 of the ‘857 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 Wetherall in light of the background knowledge of one of ordinary skill in the art, under 35 U.S.C. § 103.

It was obvious to supplement the teachings of Pfeifer96 with Wetherall. Pfeifer96 discloses application-specific “Service Gateways” and the processing of incoming “video.” Ex. 3 at 126, 118. Wetherall teaches that its “Active IP Option” technique is “a generic capability” that should be applied in particular to “application-specific service gateways” including “video gateways.” Ex. 47 at 35, 33.

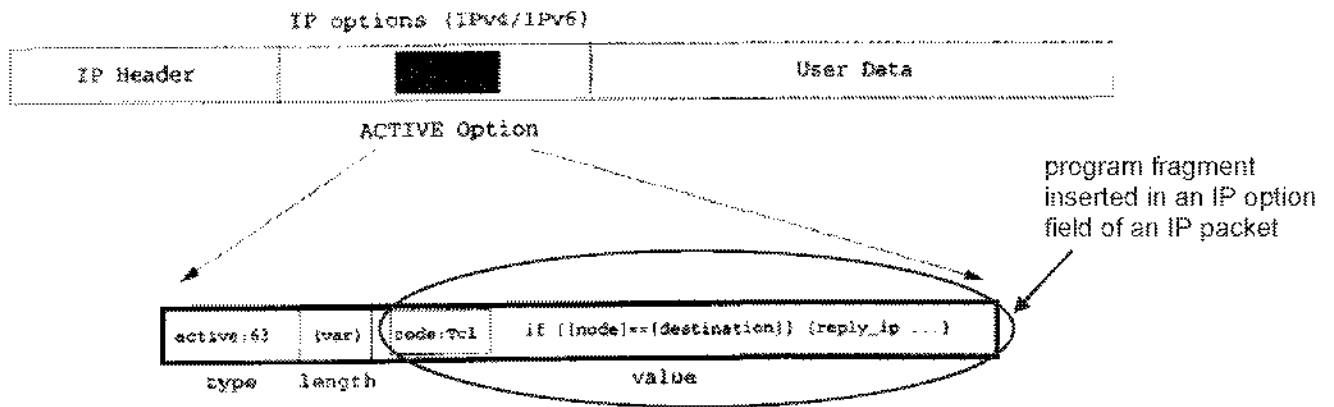
(a) Claim 1

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

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

Wetherall teaches a technique called “Active Networks,” which “break with tradition by allowing the network to perform customized computations on the user data.” Ex. 47 at 33. “For example, a user of an active network could send a customized video transcoding program to a node within the network (e.g., a router) and request that the node execute that program when processing their packets.” *Id.*

Wetherall “retrofit[s]” these “active capabilities” atop “the existing Internet” by exploiting the existing “options mechanism of the IP layer to carry program fragments.” *Id.* at 35. IP options are of flexible length and type (“generic type-length-value format of IP options”), and Wetherall simply defines a new option type “to carry program fragments, which may be encoded in a variety of languages.” *Id.* at 36.



Id. at 35 (Figure 1: “Format of the Active IP Option Field,” showing “code” in the “Tcl” programming language⁴⁰ embedded in an IP option field of a packet). Thereby, “passive packets of present day architecture” are replaced “with active ‘capsules’ – miniature programs that are executed at each router [or other device] they traverse.” *Id.* at 34-35 (“video gateways” and “application-specific service gateways” cited as other devices to support the technique).

Both the already-discussed components of Pfeifer96 and the actively-delivered components of Wetherall would perform “processing” on the packets of a particular incoming data stream (which would comprise a “message” under Implicit’s claim constructions). *E.g.*, Ex. 47 (Wetherall) at 33 (“a user of an active network could send a customized video transcoding program to a node . . . and request that the node execute that program when processing *their packets.*”) (emphasis added).

ii. “receiving a packet of the message”

Claim 1 further recites: “receiving a packet of the message and a data type of the message.” Under Implicit’s apparent claim constructions, Pfeifer96 in view of Wetherall renders obvious this element.

⁴⁰ Ex. 47 (Wetherall) at 37 (“Our first language encoding is Tcl, which is processed by a . . . Tcl interpreter” in the receiving device’s “kernel”).

In addition to the data type analysis performed by Pfeifer96 alone, Wetherall further requires that the data type of IP option field(s) in a packet be analyzed, to determine whether they contain code. *Id.* at 35-36.

iii. “dynamically identify a sequence of components”

Claim 1 further recites: “analyzing the data type of a first packet of the message to dynamically identify a sequence of components for processing a plurality of packets of the message such that the output format of the components of the sequence match the input format of the next component in the sequence, wherein analyzing the data type of the first packet of the message to dynamically identify the sequence of components includes selecting individual components to form the sequence of components after the first packet of the message is received.” Under Implicit’s apparent claim constructions, Pfeifer96 in view of Wetherall renders obvious this element.

Wetherall teaches that each packet may contain both “User data” and one or more programs in IP options field(s). *Id.* at 35 (“miniature programs,” and Figure 1 showing “User Data” plus “IP options” fields). 36 (“These fragments are . . . executed by active routers along the path taken by the datagram”).

Wetherall also teaches that “a user of an active network could send a customized video transcoding program to a node within the network (e.g., a router) and request that the node execute that program when processing their packets.” *Id.* at 33.

Pfeifer96 already employs converter components which apply “appropriate encoding/decoding” algorithms for “[c]onverting video formats.” Ex. 3 (Pfeifer96) at 113.

Applying Wetherall to Pfeifer96, a first packet of a video stream sent to Pfeifer96 could contain both video data and one or more programs for performing coding and/or other

conversions on that video stream. Ex. 3 (Pfeifer96) at 118 (Figure 9 showing incoming “video” converted to various mediums).

Either alone or as combined with other converter components identified for the message by Pfeifer96, these programs would comprise “a sequence of components for processing a plurality of packets of the message.”

The “data type” of the first packet is analyzed under Implicit’s claims constructions, *e.g.*, by analyzing the data type of the IP option field(s) to determine whether they contain code. Pfeifer96 alone also analyzes the source medium of the first packet (*e.g.*, whether it is video message), which would also comprise analyzing a “data type.” *See* Section A.B.1 (Pfeifer 102) at Claim 1(iii).

The sequence is identified “dynamically” because some of its component(s) did not even exist in the system until the first packet arrived. Indeed, it is difficult to imagine how a sequence could be identified any more “dynamically” than by obtaining one or more of the identified components from “the first packet” itself.

Other aspects of this element are discussed above. *See* Section V.C.2 (Pfeifer96 103) at Claim 1(iii).

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

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

Wetherall teaches that actively delivered programs “can leave information behind in a node,” and that this information “may be in the form of programs.” Ex. 47 at 34. Wetherall also teaches that such actively delivered programs would be stored and applied to subsequent packets

of a message. *Id.* at 33 (“a user . . . could send a customized video transcoding program to a node within the network . . . and request that the node executed that program when processing *their packets.*”) (emphasis added).

This element is discussed further above. *See* Section V.C.2 (Pfeifer96 103) at Claim 1(iv).

v. “state information”

Claim 1 finally recites: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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 Wetherall renders obvious this “state information” element

Wetherall teaches that each actively delivered component can “leave a small amount of associated state at each node along the path it traverses.” Ex. 47 at 34. “Subsequent packets can include code whose execution is dependent on this state.” *Id.*

This element is discussed further above. *See* Section V.C.2 (Pfeifer96 103) at Claim 1(v).

(b) Claim 4

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

Claim 4 recites: “A method in a computer system for processing a message, the message having a plurality of headers, the method comprising” Under Implicit’s apparent claim constructions, Pfeifer96 in view of Wetherall renders obvious this element.

In addition to the need for Pfeifer96 alone to analyze a plurality of headers of the message (*see* Section V.C.1 (Pfeifer96 102) at Claim 4(i)), Wetherall teaches that one or more IP

options headers would need to be analyzed in order to obtain the program(s) they contain. Ex. 47 at 35-36.

ii. dynamically identify a sequence of components

Claim 4 further recites: “analyzing the plurality of headers of a first packet of the message to dynamically identify a sequence of components for processing a plurality of packets of the message such that the output format of the components of the sequence match the input format of the next component in the sequence, wherein analyzing the plurality of headers of the first packet of the message to dynamically identify the sequence of components includes selecting individual components to form the sequence of components after the first packet of the message is received.” Under Implicit’s apparent claim constructions, Pfeifer96 in view of Wetherall renders obvious this element.

In addition to the need for Pfeifer96 alone to analyze a plurality of headers of the first packet to identify a sequence of components (*see* Section V.C.1 (Decasper98 102) at Claim 4(ii)), Wetherall teaches that one or more IP options headers would need to be analyzed as well, in order to obtain the program(s) they contain. Ex. 47 at 35-36. Other aspects of this element are discussed above. *See* Claim 1(iii) above.

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

Claim 4 further recites: “storing an indication of each of the identified components so that the sequence does not need to be re-identified for subsequent packets of the message.” Under Implicit’s apparent claim constructions, Pfeifer96 in view of Wetherall renders obvious this element. *See* Claim 1(iv) above.

iv. “state information”

Claim 4 finally recites: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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 Wetherall renders obvious this element. *See* Claim 1(v) above.

(c) Claim 10

Pfeifer96 in view of Wetherall renders obvious claim 10. *See* Claim 1 above.

12. Pfeifer96 in View of Wetherall, ISDN98, and Nelson Renders Obvious Claims 1, 4, and 10 Under § 103

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. As applied to the previous combination of Pfeifer96 in view of Wetherall, ISDN98 and Nelson further confirm that “a plurality of components” in a sequence would maintain “state information” across packets in the manner recited by claims 1, 4, and 10. *See* Sections V.A.11 (Pfeifer96+Wetherall) and V.A.3 (Pfeifer96+ISDN98+Nelson) above.

13. Pfeifer96 in View of Li Renders Obvious Claims 1, 4, and 10 Under § 103

The paper “Active Gateway: A Facility for Video Conferencing Traffic Control” (Exhibit 48, “Li”) by Shunge Li and Bharat Bhargava was published on February 1, 1997. It was not considered during the prosecution of the ‘857 patent.

If certain aspects recited in claims 1, 4, and 10 of the ‘857 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 Li in light of the background knowledge of one of ordinary skill in the art, under 35 U.S.C. § 103.

It was obvious to supplement the teachings of Pfeifer96 with Li. Pfeifer96 teaches a “Services Gateway[.]” for “multimedia conferencing,” and the processing of incoming “video” as controlled by adjustable “Quality of Service (QoS) parameters.” Ex. 3 at 126, 118, 115. Li

teaches an “Active Gateway” which applies “active network” technology to “video conferencing traffic and quality of service (QoS) control.” Ex. 48 at 24, 1.

(a) Claim 1

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

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

Both Pfeifer96 and Li teach that the packets of an incoming message arrive at a “gateway” where they are processed, and such a gateway would comprise “a computer system” under Implicit’s apparent claim constructions. Ex. 3 (Pfeifer96) at 23 (“Service Gateways,” *e.g.*, for “multimedia conferencing”); Ex. 48 (Li) at 10 (Figure 3: “Components of an Active Gateway” for video conferencing, including “Input Packet Queues”).

ii. “receiving a packet of the message”

Claim 1 further recites: “receiving a packet of the message and a data type of the message.” Under Implicit’s apparent claim constructions, Pfeifer96 in view of Li renders obvious this element. Pfeifer96 teaches this, and Li additionally determines a data type of a packet by determining if it is an “active network” packet containing embedded “Tcl scripts.” *See* Section V.C.1 (Pfeifer96 102) at Claim 1(ii); Ex. 48 at 7.

iii. “dynamically identify a sequence of components”

Claim 1 further recites: “analyzing the data type of a first packet of the message to dynamically identify a sequence of components for processing a plurality of packets of the message such that the output format of the components of the sequence match the input format of the next component in the sequence, wherein analyzing the data type of the first packet of the message to dynamically identify the sequence of components includes selecting individual

components to form the sequence of components after the first packet of the message is received.” Under Implicit’s apparent claim constructions, Pfeifer96 in view of Li renders obvious this element.

Implicit has characterized the “dynamically identify” 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 9 (emphasis added).

Li takes such dynamic configurability by a human administrator at run-time *for granted* as “typical” of the prior art—and extends the notion even further by permitting dynamic configuration of its “active gateways” to be performed by Tcl programs embedded in “active network” packets:

An active network isn't something that is different from current networks in its entirety For example, a typical commercial router . . . can be easily configured by *system administrators* based on their running environments. This kind of configurability is *very limited compared to the programmability the active network offers*, however, because human knowledge about when and what to configure in the router under a certain condition has not been incorporated into a program; the configuration task is *currently performed through human interaction with the router at run time*. With active network, we can program this human knowledge

into the configuration task and launch the program at the intermediate routers. Routers will no longer need the system administrators to configure them; the program does it all

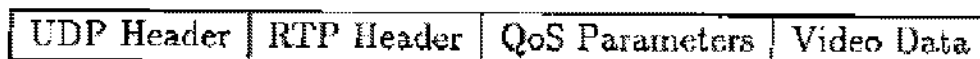
Active gateways support many control policies and scheduling policies that can be either static or dynamic or both. During initialization, an active gateway loads predefined policies. During execution of a program, a newly defined policy can be taken into effect by replacing the current policy. This *dynamic reconfigurability of policies* can be programmed in Tcl scripts. Each policy has a unique policy number and is implemented as a Tcl procedure. Therefore, a policy can be overwritten by redefining the corresponding Tcl procedure *on-the-fly*.

Ex. 48 at 3, 12 (emphasis added). Regarding these “Tcl scripts,” Li teaches they “are encapsulated in UDP datagrams, which are in turn delivered in conventional networks.” *Id.* at 7. By embedding these Tcl scripts in IP packets, Li is expressly following the teaching of the Wetherall article discussed above. *See* Ex. 48 (Li) at 3 (citing the Wetherall article (“WT96”) and observing it teaches “Tcl scripts . . . embedded in IP packets”).

The difference in emphasis between Wetherall and Li is that Li stresses these actively-delivered Tcl programs can be used to *dynamically reconfigure* the policies of the receiving device. *E.g.*, *id.* at 13 (“dynamical reconfiguration of parameters and policies”).

Regarding such dynamic reconfiguration by Tcl scripts, Li teaches certain “primitives” which “are the smallest programmable units that can perform the most basic functionality for a specific application in an active network.” *Id.* at 11. Primitives include operations on Quality of Service (“QoS”) parameters such as “network QoS (querying link latency, throughput), and application QoS (querying delay, jitter, connection information).” *Id.*

Li teaches “QoS Parameters” can be embedded directly in the headers of a packet:



Id. at 14 (Table 2: Data Encapsulation in Modified Version of NV,” where NV is a “modified” version of “a popular Internet video conferencing tool called Network Video”).

It was obvious to apply the dynamic configuration techniques of Li to Pfeifer96, because Pfeifer96 also teaches the processing of incoming “video” and “multimedia conferencing” messages, and does so based on “Quality of Service (QoS) parameters” concerning “delay,” “jitter,” “input/output data volume” (throughput), and a number of other QoS aspects. Ex. 3 (Pfeifer96) at 119-20, 118, 126.

As applied to Pfeifer96, these techniques would permit the dynamic reconfiguration of Pfeifer’s QoS parameters (or other policies) at any moment during runtime up to the first packet of an incoming message, and would even permit the QoS parameters for the message to be contained in the first packet itself. Ex. 48 (Li) at 14, 12. The QoS parameters are crucial for identifying the most appropriate sequence of conversion components for a message, and altering the QoS parameters could alter the sequence of components identified. Ex. 3 (Pfeifer96) at 120 (“Comparing different possibilities of concatenating converters for a specific task requires a complex evaluation of the quality parameters involved, performed at runtime.”). The combination of Pfeifer96 and Li would therefore “dynamically identify a sequence of components for processing a plurality of packets of the message,” under Implicit’s apparent claim constructions.

The sequence is identified by “analyzing the data type of a first packet of the message” because, *e.g.*, the sequence is based on a determination of the message’s source medium (*e.g.*, is it video?). This and other aspects of the larger claim element are discussed above. *See* Section V.C.2 (Pfeifer96 103) at Claim 1(iii).

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

Claim 1 further recites: “storing an indication of each of the identified components so that the 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. *See* Section V.C.2 (Pfeifer96 103) at Claim 1(iv).

v. “state information”

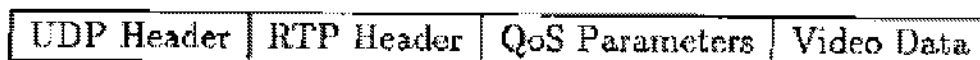
Claim 1 finally recites: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; and storing state information relating to the processing of the component with the packet for use when processing the next packet of the message.” Under Implicit’s apparent claim constructions, Pfeifer96 renders obvious this element. *See* Section V.C.2 (Pfeifer96 103) at Claim 1(v).

(b) Claim 4

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

Claim 4 recites: “A method in a computer system for processing a message, the message having a plurality of headers, the method comprising . . .” Under Implicit’s apparent claim constructions, Pfeifer96 in view of Li renders obvious this element.

In addition to Pfeifer96’s existing need to analyze a plurality of headers of the message, Li teaches that a header containing pertinent “QoS Parameters” would need to be analyzed as well.



Ex. 48 at 14 (Table 2).

ii. dynamically identify a sequence of components”

Claim 4 further recites: “analyzing the plurality of headers of a first packet of the message to dynamically identify a sequence of components for processing a plurality of packets of the message such that the output format of the components of the sequence match the input format of the next component in the sequence, wherein analyzing the plurality of headers of the first packet of the message to dynamically identify the sequence of components includes selecting individual components to form the sequence of components after the first packet of the message is received.” Under Implicit’s apparent claim constructions, Pfeifer96 in view of Li renders obvious this element.

In addition to Pfeifer96’s existing need to analyze a plurality of headers of the message to dynamically identify a sequence of components, Li teaches that a header containing pertinent “QoS Parameters” would need to be analyzed as well. Ex. 48 at 14. Other aspects of this element are discussed above. *See* Claim 1(iii).

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

Claim 4 further recites: “storing an indication of each of the identified components so that the 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. *See* Claim 1(iv) above.

iv. “state information”

Claim 4 finally recites: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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 element. *See* Claim 1(v) above.

(c) *Claim 10*

Pfeifer96 in view of Li renders obvious claim 10. *See* Claim 1 above.

14. Pfeifer96 in View of Li, ISDN98, and Nelson Renders Obvious Claims 1, 4, and 10 Under § 103

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. As applied to the previous combination of Pfeifer96 in view of Li, ISDN98 and Nelson further confirm that “a plurality of components” in a sequence would maintain “state information” across packets in the manner recited by claims 1, 4, and 10. *See* Sections V.A.13 (Pfeifer96+Li) and V.A.3 (Pfeifer96+ISDN98+Nelson) above.

15. Pfeifer96 in View of Wetherall and Li Renders Obvious Claims 1, 4, and 10 Under § 103

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.

It was obvious to supplement the teachings of Pfeifer96 with Wetherall and Li. Pfeifer96 discloses application-specific “Services Gateways” (*e.g.*, for “multimedia conferencing”), and the processing of incoming “video” as controlled by “Quality of Service (QoS) parameters.” Ex. 3 at 126, 118, 115. Wetherall teaches “Active Networks,” which is “a generic capability” that should be applied in particular to “application-specific service gateways” including “video gateways.” Ex. 47 at 35, 33. Li expressly cites Wetherall and is similarly directed to providing an “active network” capability, including through provision of an “Active Gateway” which applies “active technology to . . . video conferencing traffic and quality of service (QoS) control.” Ex. 48 (Li) at 24, 1.

The combination of these references would enable the first packet of an incoming message to contain both: (1) per **Wetherall**, one or more components for processing the message; and (2) per **Li**, “QoS Parameters” which would influence the selection of components for the converter sequence. For at least these reasons, the identification of components would be performed “dynamically,” based on the first packet of the message.

Li also teaches that by sending the system active packets containing suitable Tcl configuration scripts, the system’s controlling policies and parameters could be dynamically reconfigured at any moment during runtime up to the first packet of an incoming message. Under Implicit’s apparent claim constructions, such a system would also read the “dynamic[]” aspect of these claims.

16. Pfeifer96 in View of Wetherall, Li, ISDN98, and Nelson Renders Obvious Claims 1, 4, and 10 Under § 103

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. As applied to the previous combination of Pfeifer96 in view of Wetherall and Li, ISDN98 and Nelson further confirm that “a plurality of components” in a sequence would maintain “state information” across packets in the manner recited by claims 1, 4, and 10. *See* Sections V.A.15 (Pfeifer96+Wetherall+Li) and V.A.3 (Pfeifer96+ISDN98+Nelson) above.

17. Pfeifer96 in View of Pfeifer97 and Alam Renders Obvious Claims 1, 4, and 10 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. 3 (Pfeifer96) at 111, Ex. 12 (Pfeifer97) at 10.

As explained above, such incoming fax scenarios may clearly read on claims 1, 4, and 10. *E.g.*, Section V.C.1 (Pfeifer96 102).

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. 3 at 119, 123-24.

18. Pfeifer96 in View of Pfeifer97 and Yun Renders Obvious Claims 1, 4, and 10 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 ‘857 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, 4, and 10 of the ‘857 patent. *See, e.g.*, Section V.C.1 (Pfeifer96 102).

19. Pfeifer96 in View of Meer96, Arbanowksi96, Pfeifer97, and Franz98 Renders Obvious Claims 1, 4, and 10 Under § 103

If certain aspects recited in claims 1, 4, and 10 of the ‘857 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, Arbanowski96, Pfeifer97, and Franz98 in light of the background knowledge of one of ordinary skill in the art, 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 these documents emphasize different aspects of the iPCSS platform, collectively they provide a comprehensive picture of the system as a whole, including its design and possible uses.

20. Pfeifer96 in View of Meer96, Arbanowski96, Pfeifer97, Franz98, ISDN98, Nelson, Cox, RFC 793, Alam, and Yun Renders Obvious Claims 1, 4, and 10 Under § 103

If certain aspects recited in claims 1, 4, and 10 of the ‘857 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, Arbanowski96, Pfeifer97, Franz98, ISDN98, Nelson, Cox, RFC 793, Alam, and Yun in light of the background knowledge of one of ordinary skill in the art 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.

Claims 1, 4, and 10 recite “dynamically identify a sequence of components.”

Pfeifer96 teaches an “iPCSS” system which “dynamically generate[s]” a chain of converter components after the first packet of a message is received. Ex. 3 at 124. **Meer96**, **Arbanowski96**, **Pfeifer97**, and **Franz98** also provide pertinent details on this process.

Claims 1, 4, and 10 recite elements regarding “state information” as relating to “a plurality of components.”

Pfeifer96 teaches converter components that would maintain “state information” across packets in the manner recited by claims 1, 4, and 10: *e.g.*, components for adapting ISDN connections, and components which perform compression/decompression. **ISDN98** confirms the obviousness of employing stateful ISDN protocol adapter algorithms which would read on these elements. **Nelson** confirms the obviousness of employing stateful compression/decompression algorithms which would read on these elements.

Cox teaches an “invocation-based metering” technique which was obvious to apply to the converter components of Pfeifer96, and this technique would read on these elements.

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). **RFC 793** confirms the obviousness of employing a stateful connection algorithm in this context, which would read on these elements.

Franz98 explains that in the iPCSS system, each converter component would maintain state information across packets in a manner which would read on these elements, 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.

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

In short, there is no aspect of claims 1, 4, and 10 which was not obvious over the prior art and combinations cited herein.

D. 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 ‘857 patent.

I. Kerr Anticipates Claims 1, 4, and 10 Under § 102(e)

(a) Claim 1

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

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

Kerr discloses “[a] method in a computer system.” For example, Kerr expressly states 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). In a preferred embodiment, packets are classified into flows on the basis of five packet header fields: “IP address for the source device”; “IP address for the destination device”; “protocol type”; “port number for the source device”; and “port number for the destination device.” *Id.* at 3:55-65. *See also id.* at 2:61-3:2. Under Implicit’s apparent claim constructions, a set of packets sharing the same values for those five header fields would comprise a “message.”

ii. “receiving a packet of the message”

Claim 1 further recites: “receiving a packet of the message and a data type of the message.” Under Implicit’s apparent claim constructions, Kerr discloses this element.

Packets are classified into flows as follows:

At a step 221, the routing device 140 *receives a packet* 150.

At a step 222, the routing device 140 identifies a message flow 160 for the packet 150. In a preferred embodiment, the routing device 140 examines a header for the packet 150 and identifies the IP address for the source device 120, the IP address for the destination device 130, and the *protocol type* for the packet 150. The routing device 140 determines the *port number for the source* device 120 and the *port number for the destination* device 130 responsive to the protocol type.

Responsive to this set of information, the routing device 140 determines a flow key 310 (described with reference to FIG. 3) for the message flow 160.

Id. at 3:55-67 (emphasis added). “[P]rotocol type” would comprise a data type under Implicit’s apparent claim constructions. Additionally, because the source and/or destination port number is typically “a standard port number” indicating the application above (*e.g.*, “FTP . . . TELNET . . . an internet telephone protocol, or an internet video protocol such as the ‘CUSeeMe’ protocol”), it would also comprise a data type under Implicit’s apparent claim constructions. *Id.* at 3:5-14.

iii. “dynamically identify a sequence of components”

Claim 1 further recites: “analyzing the data type of a first packet of the message to dynamically identify a sequence of components for processing a plurality of packets of the message such that the output format of the components of the sequence match the input format of the next component in the sequence, wherein analyzing the data type of the first packet of the message to dynamically identify the sequence of components includes selecting individual components to form the sequence of components after the first packet of the message is received.” Under Implicit’s apparent claim constructions, Kerr discloses this element.

Implicit has characterized the “dynamically identify” 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 9 (emphasis added).

Kerr discloses “dynamically identify” 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 identify a sequence of components” under Implicit’s apparent claim construction. *See* Section IV. 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 sequence of components is identified by “analyzing the data type of a first packet of the message” because the sequence is tailored per flow, and flows are classified based on multiple header fields which would comprise a data type, including “protocol type” and “port

number for the source” and/or “port number for the destination.” *Id.* at 4:20-34, 3:58-67, 3:9-14. *See also* Claim 1(ii) above.

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. 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 identify” element as disclosed in Kerr (under Implicit’s apparent claim construction) also “includes selecting individual components to form the sequence of components after the first packet of the message 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. 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 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 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.

v. “state information”

Claim 1 finally recites: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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. 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 4

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

Claim 4 recites: “A method in a computer system for processing a message, the message having a plurality of headers, the method comprising” Under Implicit’s apparent claim constructions, Kerr discloses this element.

As explained above, as packets arrive, they are classified into flows based on the following packet header fields: “IP address for the source device”; “IP address for the destination device”; “protocol type”; “port number for the source device”; and “port number for the destination device.” *Id.* at 3:55-65. The first three fields are found in an IP packet’s layer 3 header, and the final two in its layer 4 header (*e.g.*, TCP or UDP).⁴¹

Other aspects of this claim element are discussed above. *See* Claim 1(i) above.

ii. “dynamically identify a sequence”

Claim 4 further recites: “analyzing the plurality of headers of a first packet of the message to dynamically identify a sequence of components for processing a plurality of packets of the message such that the output format of the components of the sequence match the input format of the next component in the sequence, wherein analyzing the plurality of headers of the first packet of the message to dynamically identify the sequence of components includes selecting individual components to form the sequence of components after the first packet of the

⁴¹ *See, e.g.*, Ex. 41 (RFC 791) (IP Specification) (1981) at 11 (“Source Address”; “Destination Address”; “Protocol”); Ex. 9 (RFC 793) (TCP Specification) (1981) at 15 (“Source Port”; “Destination Port”). These references are cited in this context solely to help explain Kerr. *See* MPEP § 2205.

message is received.” Under Implicit’s apparent claim constructions, Kerr discloses this element.

Regarding the limitation “*analyzing the plurality of headers of a first packet of the message to . . . identify a sequence of components,*” Kerr teaches that the sequence is tailored per flow, and flows are classified based on multiple header fields which would comprise a data type, including “protocol type” and “port number for the source” and/or “port number for the destination.” Ex. 15 at 4:20-34, 3:58-67, 3:9-14. *See also* Claim 1(ii) above.

Other aspects of this claim element are discussed above. *See* Claim 1(iii) above.

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

Claim 4 further recites: “storing an indication of each of the identified components so that the sequence does not need to be re-identified for subsequent packets of the message.” Under Implicit’s apparent claim constructions, Kerr discloses this element. *See* Claim 1(iv) above.

iv. “state information”

Claim 4 finally recites: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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 element. *See* Claim 1(v) above.

(c) Claim 10

i. “A computer readable storage medium”

Claim 10 recites in pertinent part: “A computer readable storage medium, other than a data transmission medium, containing instructions for processing packets of a message, the instructions comprising at least one computer-executable module configured to . . .” 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 storage medium, other than a data transmission medium”: *e.g.*, from a hard disk in the device. Ex. 15 at 2:53-54.

Other aspects of this claim element are discussed above. *See* Claim 1(i) above.

ii. Other claim elements

The remaining elements of claim 10 are also disclosed by Kerr. *See* Claim 1 above.

2. Kerr Renders Obvious Claims 1, 4, and 10 Under § 103

If certain aspects recited in claims 1, 4, and 10 of the ‘857 patent are not deemed to be disclosed or inherent over Kerr, then the inclusion of those aspects certainly would be obvious over Kerr in light of the background knowledge of one of ordinary skill in the art, under 35 U.S.C. § 103.

(a) Claim 1

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

Claim 1 recites: “A method in a computer system for processing packets of a message, 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. Other aspects of this claim element are discussed above. *See* Section V.D.1 (Kerr 102) at Claim 1(i) above.

ii. “receiving a packet of the message”

Claim 1 further recites: “receiving a packet of the message and a data type of the message.” Under Implicit’s apparent claim constructions, Kerr renders obvious this element.

As packets arrive, they are classified into flows on the basis of five header fields including “protocol type,” “port number for the source,” and “port number for the destination.” Ex. 15 at 3:55-67. “[P]rotocol type” would comprise a data type under Implicit’s apparent claim constructions. Additionally, because the source and/or destination port number is typically “a standard port number” indicating the application above (*e.g.*, “FTP . . . TELNET . . . an internet telephone protocol, or an internet video protocol such as the ‘CUSeeMe’ protocol”), it was obvious this would comprise “a data type” under Implicit’s apparent claim constructions. *Id.* at 3:5-14.

iii. “dynamically identify a sequence of components”

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

Regarding the limitation “sequence of components,” it was obvious that the various distinct operations performed on the packets of a particular flow would be organized as distinct components. This is an elementary design technique, well-understood by those of ordinary skill in the art. Indeed, Kerr expressly mentions its rewrite operation would be implemented as such. Ex. 15 at 4:56-57 (“the flow cache includes a pointer to a rewrite *function*”) (emphasis added).

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* Ex. 15 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 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).

Other aspects of this claim element are discussed above. *See* Section V.D.1 (Kerr 102) at Claim 1(iii) above.

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

Claim 1 further recites: “storing an indication of each of the identified components so that the sequence does not need to be re-identified for subsequent packets of the message.” Under Implicit’s apparent claim constructions, Kerr discloses this element. *See* Section V.D.1 (Kerr 102) at Claim 1(iv) above.

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” Ex. 15 at 4:12-16 (emphasis added). Since it was obvious to implement the various distinct operations comprising the proper treatment as distinct components, “enter[ing] information regarding such treatment” would comprise storing an indication of the corresponding components. *See* Claim 1(iii) above. Indeed, Kerr expressly mentions storing an indication of the rewrite function component in the data structure. *Id.* at 4:56-57 (“the flow cache includes a pointer to a rewrite *function*”) (emphasis added).

v. “state information”

Claim 1 finally recites: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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. 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 of common techniques for implementing an encryption algorithm which “MUST” be supported by the security architecture for the Internet Protocol: *i.e.*, the “ESP DES-CBC” algorithm described in RFC 1829.⁴² In order to apply this required 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). Ex. 27 (RFC 1829) at 1. RFC 1829 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 common technique to implement the encryption component of Kerr. Doing so would clearly entail, for each packet, retrieving the previous counter value, applying it to encrypt the packet, 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 4

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

Claim 4 recites: “A method in a computer system for processing a message, the message having a plurality of headers, the method comprising” Under Implicit’s apparent claim constructions, Kerr discloses this element. *See* Section V.D.1 (Kerr 102) at Claim 4(i) above.

⁴² *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 Transform”). RFC 1825 and RFC 1829 are cited in this section solely to help explain Kerr. *See* MPEP § 2205.

ii. “dynamically identify a sequence”

Claim 4 further recites: “analyzing the plurality of headers of a first packet of the message to dynamically identify a sequence of components for processing a plurality of packets of the message such that the output format of the components of the sequence match the input format of the next component in the sequence, wherein analyzing the plurality of headers of the first packet of the message to dynamically identify the sequence of components includes selecting individual components to form the sequence of components after the first packet of the message is received.” Under Implicit’s apparent claim constructions, Kerr renders obvious this element.

Regarding the limitation “analyzing the plurality of headers of a first packet of the message to dynamically identify a sequence,” Kerr discloses this element. *See* Section V.D.1 (Kerr 102) at Claim 4(ii) above. Other aspects of this claim element are discussed above. *See* Claim 1(iii) above.

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

Claim 4 further recites: “storing an indication of each of the identified components so that the 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. *See* Claim 1(iv) above.

iv. “state information”

Claim 4 finally recites: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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 element. *See* Claim 1(v) above.

(c) *Claim 10*

i. *“A computer readable storage medium”*

Claim 10 recites in pertinent part: “A computer readable storage medium, other than a data transmission medium, containing instructions for processing packets of a message, the instructions comprising at least one computer-executable module configured to” Under Implicit’s apparent claim constructions, Kerr renders obvious this element.

Kerr teaches its invention may be performed by “a general purpose processor operating under program control,” and it was obvious that such a controlling program would be loaded from a “computer readable storage medium, other than a data transmission medium”: *e.g.*, from a hard disk in the device. Ex. 15 at 2:53-54. Other aspects of this claim element are discussed above. *See* Claim 1(i) above.

ii. *Other claim elements*

The remaining elements of claim 10 are also rendered obvious by Kerr. *See* Claim 1 above.

3. Kerr in View of NetFlow Renders Obvious Claims 1, 4, and 10 Under § 103

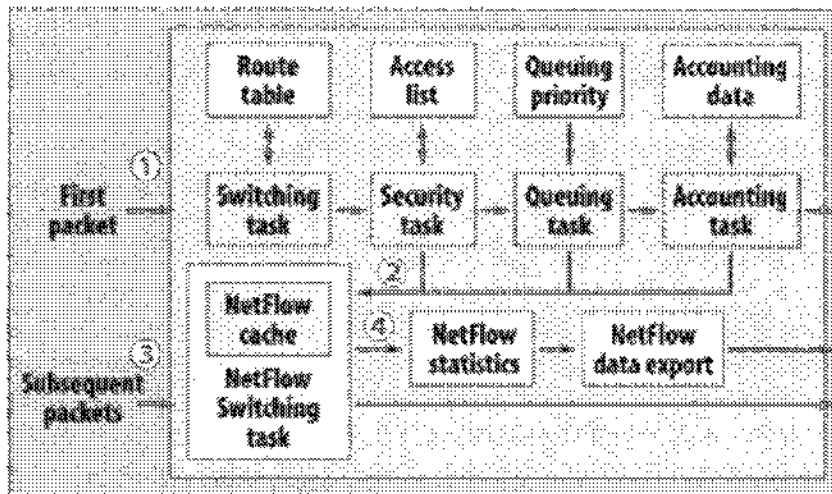
The article “Cisco NetFlow Switching speeds traffic routing” (Exhibit 16, “NetFlow”) by Stephen Lawson was published on July 7, 1997 in the publication InfoWorld. NetFlow was not considered during the prosecution of the ‘857 patent.

If certain aspects recited in claims 1, 4, and 10 of the ‘857 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 NetFlow in light of the background knowledge of one of ordinary skill in the art, 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, 4, and 10, 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, 4, and 10 Under § 103

The specification RFC 1825 (“Security Architecture for the Internet Protocol”) (Exhibit 26, “RFC 1825”) by R. Atkinson was published in August 1995. The specification RFC 1829 (Exhibit 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 ‘857 patent.

If certain aspects recited in claims 1, 4, and 10 of the ‘857 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 in light of the background knowledge of one of ordinary skill in the art, 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

Claim 1 recites in pertinent part: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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 this “state information” 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 (RFC 1825) 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 that need not be applied to the same packet, it was obvious to implement them as distinct components. *See, e.g., id.* at 8

(“The two IP security mechanisms [authentication and encryption] may be used together or separately”).

RFC 1825 explains that various forms of state information would be maintained by these encryption and authentication 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. 26 (RFC 1825) at 5-6. Obvious implementations to maintain this state information would read on this “state information” claim element, under Implicit’s apparent claim constructions. For example, both the encryption and authentication component(s) 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 algorithm 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 this required 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). 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 “common, acceptable” counter technique to an encryption component of Decasper98. Doing so would entail, for each packet, retrieving the previous counter value, applying it to encrypt the packet, 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.

To summarize, Kerr in view of RFC 1825 and RFC 1829 renders obvious distinct components for encryption and authentication which would maintain state information across packets in the manner recited by claim 1. Additional stateful Kerr components are discussed above. *See* Section V.D.2 (Kerr 103) at Claim 1(v).

(b) Claim 4

Claim 4 recites in pertinent part: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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 this element. *See* Claim 1 above.

(c) Claim 10

Claim 10 recites in pertinent part: “for each of a plurality of components in the identified sequence: perform the processing of each packet by the identified component; and store 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 this element. *See* Claim 1 above.

5. Kerr in View of Bellare97 and Bellare95 Renders Obvious Claims 1, 4, and 35 Under § 103

The article “A Concrete Security Treatment of Symmetric Encryption” (Exhibit 17, “Bellare97”) by M. Bellare *et al.* was published in 1997. The article “XOR MACs: New Methods for Message Authentication Using Finite Pseudorandom Functions” (Exhibit 18, “Bellare95”) by M. Bellare et al. was published in 1995. Neither was considered during the prosecution of the ‘857 patent.

If certain aspects recited in claims 1, 4, and 10 of the ‘857 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 in light of the background knowledge of one of ordinary skill in the art, 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 (Kerr) 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

Claim 1 recites in pertinent part: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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 “state information” claim element.

As explained above, it was obvious that each of the various distinct operations performed by Kerr on the packets of a particular flow (including its encryption operation) would be organized as distinct components. *See* Section V.D.2 (Kerr 103) at Claim 1(iii) above.

Bellare95 teaches another distinct operation that would advantageous to apply to the packets of a flow—“Authentication”—and it was obvious that this operation would be provided by a distinct component as well. Ex. 18 (Bellare95) 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 (Kerr) at 4:30-32.

Regarding the implementation of the distinct encryption component, 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.

Regarding the implementation of the distinct authentication 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_i 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.

To summarize, Kerr in view of Bellare97 and Bellare95 renders obvious distinct components for encryption and authentication which would maintain state information across packets in the manner recited by claim 1. Additional stateful Kerr components are discussed above. *See* Section V.D.2 (Kerr 103) at Claim 1(v).

(b) Claim 4

Claim 4 recites in pertinent part: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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. *See* Claim 1 above.

(c) Claim 10

Claim 10 recites in pertinent part: “for each of a plurality of components in the identified sequence: perform the processing of each packet by the identified component; and store 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. *See* Claim 1 above.

6. Kerr in View of IBM96 Renders Obvious Claims 1, 4, and 10 Under § 103

The book “Local Area Network Concepts and Products: Routers and Gateways” (Exhibit 19, “IBM96”) was published by IBM in May 1996. IBM96 was not considered during prosecution of the ‘857 patent.

If certain aspects recited in claims 1, 4, and 10 of the ‘857 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 in light of the background knowledge of one of ordinary skill in the art, 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

Claim 1 recites in pertinent part: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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 this “state information” claim 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, 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. *I.g.*, Ex. 15 at 4:30-31.

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 this “state information” claim element.⁴³ 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.⁴⁴

To summarize, Kerr in view of IBM96 renders obvious a compression component employing an “adaptive” algorithm which would maintain state information across packets in the manner recited by claim 1. Additional stateful Kerr components are discussed above. *See* Section V.D.2 (Kerr 103) at Claim 1(v).

(b) Claim 4

Claim 4 recites in pertinent part: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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 this element. *See* Claim 1 above.

⁴³ *See, e.g.*, Ex. 5 (Nelson) (“The Data Compression Book”) (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.

⁴⁴ *See, e.g.*, Ex. 5 (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.

(c) Claim 10

Claim 10 recites in pertinent part: “for each of a plurality of components in the identified sequence: perform the processing of each packet by the identified component; and store 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 this element. *See* Claim 1 above.

7. Kerr in View of IBM96 and Nelson Renders Obvious Claims 1, 4, and 10 Under § 103

The treatise “The Data Compression Book” (Exhibit 5, “Nelson”) by Mark Nelson *et al.* was published on November 6, 1995. Nelson was not considered during the prosecution of the ‘857 patent.

If certain aspects recited in claims 1, 4, and 10 of the ‘857 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 in light of the background knowledge of one of ordinary skill in the art, under 35 U.S.C. § 103.

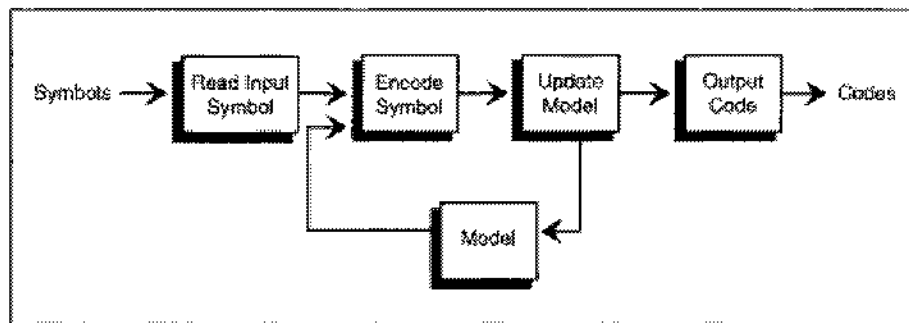
It was obvious to supplement the teachings of Kerr and IBM96 with Nelson, because IBM96 discloses compression operations performed by routers, and Nelson teaches specific compression algorithms which might be used.

(a) Claim 1

Claim 1 recites in pertinent part: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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” claim 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 adaptive algorithms would have been an obvious choice for compression components.

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.

As observed above, Nelson teaches a number of lossless, adaptive compression algorithms in Chapters 1 to 9 which would have been obvious choices for a compression component of Kerr. *See id.* at 9 (“All of the compression techniques discussed through chapter 9 are ‘lossless’”).

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

To summarize, Kerr in view of IBM96 and Nelson renders obvious a compression component employing an “adaptive” algorithm which would maintain state information across packets in the manner recited by claim 1. Additional stateful Kerr components are discussed above. *See Section V.D.2 (Kerr 103) at Claim 1(v).*

(b) Claim 4

Claim 4 recites in pertinent part: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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 element. *See* Claim 1 above.

(c) Claim 10

Claim 10 recites in pertinent part: “for each of a plurality of components in the identified sequence: perform the processing of each packet by the identified component; and store 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 element. *See* Claim 1 above.

8. Kerr in View of RFC 1825, RFC 1829, Bellare97, Bellare95, IBM96, and Nelson Renders Obvious Claims 1, 4, and 10 Under § 103

If certain aspects recited in claims 1, 4, and 10 of the ‘857 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 in light of the background knowledge of one of ordinary skill in the art, 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, and it anticipates and renders obvious all elements of claims 1, 4, and 10.

Claims 1, 4, and 10 recite elements regarding “state information” as relating to “a plurality of components.”

Kerr teaches, *e.g.*, encryption, packet re-write, and various other “special treatment” applied to the packets of specific flows. *I.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.

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

Since Kerr teaches that its various operations are applied in a manner tailored to each flow (*e.g.*, *id.* at 4:12-20), it was obvious that any two of more of these stateful components (encryption, authentication, compression) would be applied to a particular flow. This was especially obvious since all three of these 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. Moreover, Kerr teaches additional stateful components which read on these “state information” claim elements, and it was also obvious for any of these to be applied to a particular flow as well. *See* Sections V.B.1 (Kerr 102) and V.B.2 (Kerr 103) above.

In short, there is no aspect of claims 1, 4, and 10 which was not obvious over the prior art and combinations cited herein.

9. Kerr in View of Fraser Renders Obvious Claims 1, 4, and 10 Under § 103

The publication “DTE Firewalls: Phase Two Measurement and Evaluation Report” (Exhibit 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 ‘857 patent.

If certain aspects recited in claims 1, 4, and 10 of the '857 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 in light of the background knowledge of one of ordinary skill in the art, under 35 U.S.C. § 103.

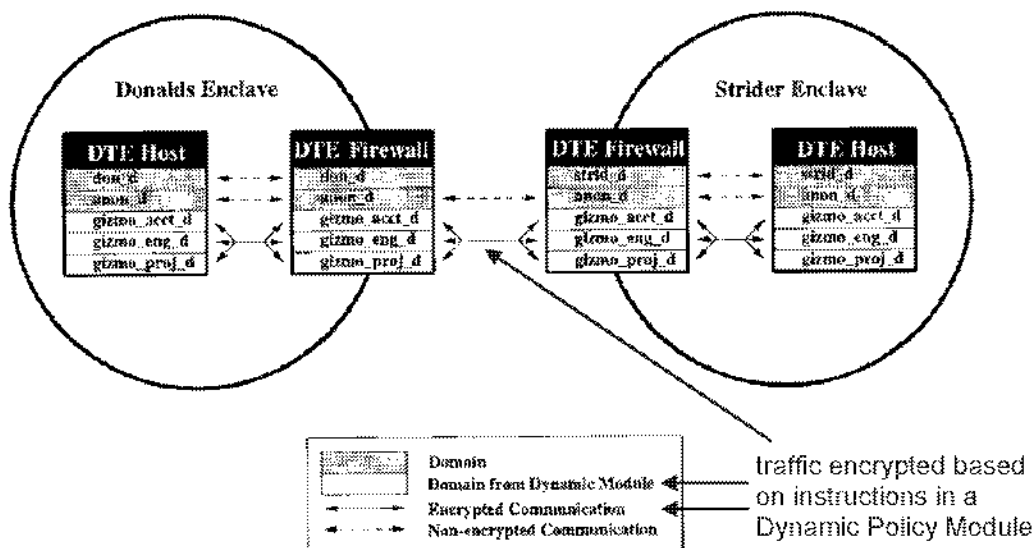
It was obvious to supplement the teachings of Kerr with Fraser because Kerr teaches a general flow-based architecture for routers and firewalls (*e.g.*, Ex. 15 at 4:12-48), and Fraser teaches a technique for enhancing the dynamic configurability of such an architecture.

(a) Claim 1

Claim 1 recites in pertinent part: “analyzing the data type of a first packet of the message to dynamically identify a sequence of components for processing a plurality of packets of the message such that the output format of the components of the sequence match the input format of the next component in the sequence, wherein analyzing the data type of the first packet of the message to dynamically identify the sequence of components includes selecting individual components to form the sequence of components after the first packet of the message is received.” Under Implicit’s apparent claim constructions, Kerr in view Fraser renders obvious this element.

Kerr alone renders obvious these elements. *See* Section V.D.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 framework⁴⁵ for avoiding any “undesirable and impractical” need to reboot the Decasper98 device under any circumstances. *See id.* at 9. Kerr was an 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 (Kerr) 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

⁴⁵ 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, e.g.*, Ex. 15 (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”).

(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 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 identify a sequence of components for processing a plurality of packets of the message.”

(b) Claim 4

Claim 4 recites in pertinent part: “analyzing the plurality of headers of a first packet of the message to dynamically identify a sequence of components for processing a plurality of packets of the message such that the output format of the components of the sequence match the input format of the next component in the sequence, wherein analyzing the plurality of headers of the first packet of the message to dynamically identify the sequence of components includes selecting individual components to form the sequence of components after the first packet of the message is received.” Under Implicit's apparent claim constructions, Kerr in view of Fraser renders obvious this element. *See* Claim 1 above.

(c) Claim 10

Claim 10 recites in pertinent part: “analyze the data type of a first packet of the message to dynamically identify a sequence of components for processing a plurality of packets of the message such that the output format of the components of the sequence match the input format of the next component in the sequence, wherein analyzing the data type of the first packet of the message to dynamically identify the sequence of components includes selecting individual

components to form the sequence of components after the first packet of the message is received.” Under Implicit’s apparent claim constructions, Kerr in view of Fraser renders obvious this element. *See* Claim 1 above.

10. Kerr in View of Fraser, Bellare97, and Bellare95 Renders Obvious Claims 1, 4, and 10 Under § 103

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. As applied to the previous combination of Kerr in view of Fraser, Bellare97 and Bellare95 further confirm that “a plurality of components” in a sequence would maintain “state information” across packets in the manner recited by claims 1, 4, and 10. *See* Sections V.B.9 (Kerr+Fraser) and V.B.5 (Kerr+Bellare97+Bellare95) above.

11. Kerr in View of Bellissard Renders Obvious Claims 1, 4, and 10 Under § 103

The article “Dynamic Reconfiguration of Agent-Based Applications” (Exhibit 23, “Bellissard”) by Luc Bellissard *et al.* was published by September 10, 1998. Bellissard was not considered during prosecution of the ‘857 patent.

If certain aspects recited in claims 1, 4, and 10 of the ‘857 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 in light of the background knowledge of one of ordinary skill in the art, 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

Claim 1 recites in pertinent part: “analyzing the data type of a first packet of the message to dynamically identify a sequence of components for processing a plurality of packets of the message such that the output format of the components of the sequence match the input format of the next component in the sequence, wherein analyzing the data type of the first packet of the message to dynamically identify the sequence of components includes selecting individual components to form the sequence of components after the first packet of the message is received.” Under Implicit’s apparent claim constructions, Kerr in view Bellissard renders obvious this element.

Kerr alone renders obvious this element. *See* Section V.D.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 “plugin” modules 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 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 plugin, and so on. Using the Bellissard technique, such a plugin 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 the plugin 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 identify a sequence of components for processing a plurality of packets of the message.”

(b) Claim 4

Claim 4 recites in pertinent part: “analyzing the plurality of headers of a first packet of the message to dynamically identify a sequence of components for processing a plurality of packets of the message such that the output format of the components of the sequence match the input format of the next component in the sequence, wherein analyzing the plurality of headers of the first packet of the message to dynamically identify the sequence of components includes selecting individual components to form the sequence of components after the first packet of the message is received.” Under Implicit’s apparent claim constructions, Kerr in view of Bellissard renders obvious this element. *See* Claim 1 above.

(c) Claim 10

Claim 10 recites in pertinent part: “analyze the data type of a first packet of the message to dynamically identify a sequence of components for processing a plurality of packets of the message such that the output format of the components of the sequence match the input format of

the next component in the sequence, wherein analyzing the data type of the first packet of the message to dynamically identify the sequence of components includes selecting individual components to form the sequence of components after the first packet of the message is received.” Under Implicit’s apparent claim constructions, Kerr in view of Bellissard renders obvious this element. *See* Claim 1 above.

12. Kerr in View of Bellissard, Bellare97, and Bellare95 Renders Obvious Claims 1, 4, and 10 Under § 103

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. As applied to the previous combination of Kerr in view of Bellissard, Bellare97 and Bellare95 further confirm that “a plurality of components” in a sequence would maintain “state information” across packets in the manner recited by claims 1, 4, and 10. *See* Sections V.B.11 (Kerr+Bellissard) and V.B.5 (Kerr+Bellare97+Bellare95) above.

13. Kerr in View of Wetherall Renders Obvious Claims 1, 4, and 10 Under § 103

The article “The Active IP Option” (Exhibit 47, “Wetherall”) by David J. Wetherall and David L. Tennenhouse was published by September 11, 1996. It was not considered during the prosecution of the ‘857 patent.

If certain aspects recited in claims 1, 4, and 10 of the ‘857 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 Wetherall in light of the background knowledge of one of ordinary skill in the art, under 35 U.S.C. § 103.

It was obvious to supplement the teachings of Kerr with Wetherall because Kerr teaches a general flow-based architecture for routers, and Wetherall teaches that its “Active IP Option”

technique is “a generic capability” that should be applied to “routers” and “[f]lows.” Ex. 15 (Kerr) at 4:12-18; Ex. 47 (Wetherall) at 34-35.

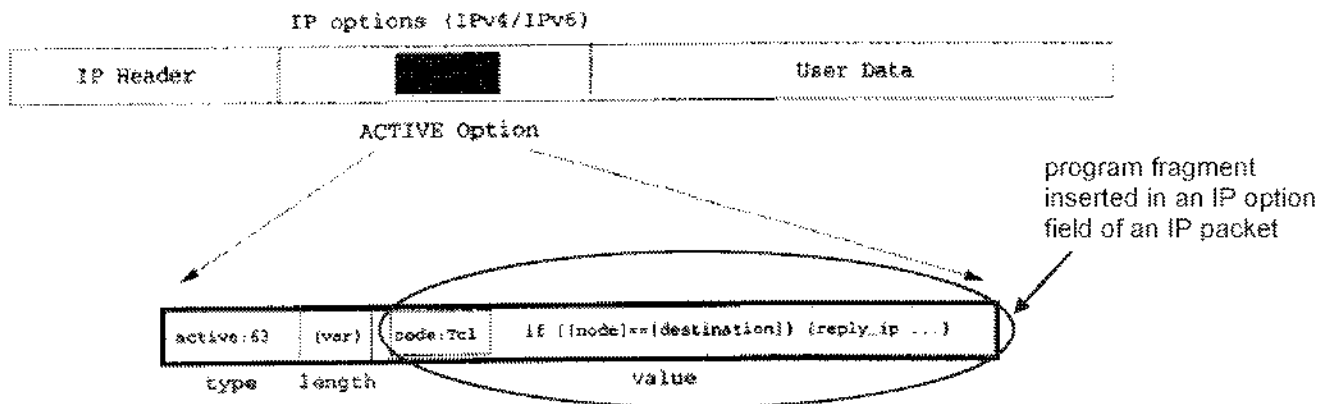
(a) Claim 1

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

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

Wetherall teaches an approach called “Active Networks,” which “break with tradition by allowing the network to perform customized computations on the user data.” Ex. 47 at 33. “For example, a user of an active network could send a customized video transcoding program to a node within the network (e.g., a router) and request that the node execute that program when processing their packets.” *Id.*

Wetherall “retrofit[s]” these “active capabilities” atop “the existing Internet” by exploiting the existing “options mechanism of the IP layer to carry program fragments.” *Id.* at 35. IP options are of flexible length and type (“generic type-length-value format of IP options”), and Wetherall simply defines a new option type “to carry program fragments, which may be encoded in a variety of languages.” *Id.* at 36.



Id. at 35 (Figure 1: “Format of the Active IP Option Field,” showing “code” embedded in an IP option field of a packet). Thereby, “passive packets of present day architecture” are replaced “with active ‘capsules’ – miniature programs that are executed at each router they traverse.” *Id.* at 34-35.

Both the already-discussed components of Kerr and the actively-delivered components of Wetherall would perform “processing” on the packets of a particular flow. *E.g.*, Ex. 47 (Wetherall) at 33 (“a user of an active network could send a customized video transcoding program to a node . . . and request that the node execute that program when processing *their packets.*”) (emphasis added).

ii. “receiving a packet of the message”

Claim 1 further recites: “receiving a packet of the message and a data type of the message.” Under Implicit’s apparent claim constructions, Kerr in view of Wetherall renders obvious this element.

In addition to the data type analysis performed by Kerr alone, Wetherall further requires that the data type of IP option field(s) in a packet be analyzed, to determine whether they contain code. *Id.* at 35-36.

iii. “dynamically identify a sequence of components”

Claim 1 further recites: “analyzing the data type of a first packet of the message to dynamically identify a sequence of components for processing a plurality of packets of the message such that the output format of the components of the sequence match the input format of the next component in the sequence, wherein analyzing the data type of the first packet of the message to dynamically identify the sequence of components includes selecting individual components to form the sequence of components after the first packet of the message is

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

Wetherall teaches that each packet may contain both “User data” and one or more programs in IP options field(s). *Id.* at 35 (“miniature programs,” and Figure 1 showing “User Data” plus “IP options” fields). 36 (“These fragments are . . . executed by active routers along the path taken by the datagram”).

Wetherall also teaches that “a user of an active network could send a customized video transcoding program to a node within the network (e.g., a router) and request that the node execute that program when processing their packets.” *Id.* at 33.

As applied to Kerr, which already processes incoming video, the first packet of a video flow could contain one or more programs to be used for performing coding and/or other operations on the packets of the flow. Ex. 15 (Kerr) at 3:13-14 (flow conveying “an internet video protocol such as the ‘CUSeeMe’ protocol”).

Either alone or as combined with other components identified for the flow by Kerr, these programs would comprise “a sequence of components for processing a plurality of packets of the message.”

The “data type” of the first packet is analyzed under Implicit’s claims constructions, e.g., by analyzing the data type of the IP option field(s) to determine whether they contain code. Kerr alone also analyzes other fields of the first packet which would comprise a “data type,” as explained above. *See* Section V.D.1 (Kerr 102) at Claim 1(iii).

The sequence is identified “dynamically” because some of its component(s) did not even exist in the system until the first packet arrived. Indeed, it is difficult to imagine how a sequence

could be identified any more “dynamically” than by obtaining one or more of the identified components from “the first packet” itself.

Other aspects of this element are discussed above. *See* Section V.D.2 (Kerr 103) at Claim 1(iii).

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

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

Wetherall teaches that actively delivered programs “can leave information behind in a node,” and that this information “may be in the form of programs.” Ex. 47 at 34. Wetherall also teaches that such actively delivered programs would be stored and applied to subsequent packets of a message. *Id.* at 33 (“a user . . . could send a customized video transcoding program to a node within the network . . . and request that the node executed that program when processing *their packets.*”) (emphasis added).

This element is discussed further above. *See* Section V.D.2 (Kerr 103) at Claim 1(iv).

v. “state information”

Claim 1 finally recites: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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 Wetherall renders obvious this “state information” element

Wetherall teaches that each actively delivered component can “leave a small amount of associated state at each node along the path it traverses.” Ex. 47 at 34. “Subsequent packets can include code whose execution is dependent on this state.” *Id.*

This element is discussed further above. *See* Section V.D.2 (Kerr 103) at Claim 1(v).

(b) Claim 4

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

Claim 4 recites: “A method in a computer system for processing a message, the message having a plurality of headers, the method comprising” Under Implicit’s apparent claim constructions, Kerr in view of Wetherall renders obvious this element.

In addition to the need for Kerr alone to analyze a plurality of headers of the message (*see* Section V.D.1 (Kerr 102) at Claim 4(i)), Wetherall teaches that one or more IP options headers would need to be analyzed in order to obtain the program(s) they contain. Ex. 47 at 35-36.

ii. “dynamically identify a sequence of components”

Claim 4 further recites: “analyzing the plurality of headers of a first packet of the message to dynamically identify a sequence of components for processing a plurality of packets of the message such that the output format of the components of the sequence match the input format of the next component in the sequence, wherein analyzing the plurality of headers of the first packet of the message to dynamically identify the sequence of components includes selecting individual components to form the sequence of components after the first packet of the message is received.” Under Implicit’s apparent claim constructions, Kerr in view of Wetherall renders obvious this element.

In addition to the need for Kerr alone to analyze a plurality of headers of the first packet to identify a sequence of components (*see* Section V.D.1 (Kerr 102) at Claim 4(ii)), Wetherall teaches that one or more IP options headers would need to be analyzed as well, in order to obtain the program(s) they contain. Ex. 47 at 35-36. Other aspects of this element are discussed above. *See* Claim 1(iii) above.

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

Claim 4 further recites: “storing an indication of each of the identified components so that the sequence does not need to be re-identified for subsequent packets of the message.” Under Implicit’s apparent claim constructions, Kerr in view of Wetherall renders obvious this element. *See* Claim 1(iv) above.

iv. “state information”

Claim 4 finally recites: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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 Wetherall renders obvious this element. *See* Claim 1(v) above.

(c) *Claim 10*

Kerr in view of Wetherall renders obvious claim 10. *See* Claim 1 above.

14. Kerr in View of Wetherall, Bellare97, and Bellare95 Renders Obvious Claims 1, 4, and 10 Under § 103

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. As applied to the previous combination of Kerr in view of Wetherall, Bellare97 and Bellare95 further confirm that “a plurality of components” in a sequence would maintain “state information” across packets in the manner recited by claims 1, 4, and 10. *See* Sections V.B.13 (Kerr+Wetherall) and V.B.4 (Kerr+Bellare97+Bellare95) above.

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

If certain aspects recited in claims 1, 4, and 10 of the '857 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, Fraser, Bellissard, and Wetherall in light of the background knowledge of one of ordinary skill in the art 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, and it anticipates and renders obvious all elements of claims 1, 4, and 10.

Claims 1, 4, and 10 recite elements regarding "state information" as relating to "a plurality of components."

Kerr teaches, *e.g.*, encryption, packet re-write, and various other "special treatment" applied 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.

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

confirms the obviousness of employing stateful compression algorithms which would read on these elements.

Since Kerr teaches that its various operations are applied in a manner tailored to each flow (*e.g.*, *id.* at 4:12-20), it was obvious that any two or more of these stateful components (encryption, authentication, compression) would be applied to a particular flow. This was especially obvious since all three of these 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. Moreover, Kerr teaches additional stateful components which read on these “state information” claim elements, and it was also obvious for any of these to be applied to a particular flow as well. *See* Sections V.B.1 (Kerr 102) and V.B.2 (Kerr 103) above.

Claims 1, 4, and 10 recite “dynamically identify a 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”).

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.

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.

Wetherall teaches dynamically delivering new component(s) in the first packet of a message, which would also read on these “dynamic[.]” claim elements. Indeed, it is difficult to imagine how a sequence could be identified any more “dynamically” than by obtaining one or more of the identified components from the first packet itself.

In short, there is no aspect of claims 1, 4, and 10 which was not obvious over the prior art and combinations cited herein.

16. Kerr in View of Checkpoint and Shwed Renders Obvious Claims 1, 4, and 10 Under § 103

The paper “Checkpoint Firewall-1 White Paper, Version 2.0” (Exhibit 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. (Exhibit 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 ‘857 patent.

If certain aspects recited in claims 1, 4, and 10 of the ‘857 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, in light of the background knowledge of one of ordinary skill in the art, 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 claimed priority date of the

'857 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":

RULE BASE EDITOR : CORPORATE

FILE | RULE | FILTER | ROUTER | UTILITES | PROPERTIES | TUTORIAL

WINDOWS: NETWORK OBJECTS SERVICES SYSTEM VIEW LOG VIEWER

| NO. | SOURCE | DESTINATION | SERVICES | ACTION | TRACK | INSTALL ON |
|-----|--|--------------------------|-------------------|----------------|------------|-----------------|
| 1 | ANY ☉ | MAILSERVERS ☉ | SMTP | ACCEPT ☒ | | GATEWAYS GW |
| 2 | <input type="checkbox"/> CEO <input type="checkbox"/> CFO | FINANCE ☉ | ANY ☉ | DROP STOP | ALERT ☘ | GATEWAYS GW |
| 3 | TRUSTEDPARTIES ☉ | INTERNAL ☉ | TALK RSTAT TELNET | ACCEPT ☒ | | DST ⊕ |
| 4 | INTERNAL ☉ | ANY ☉ | ANY ☉ | ACCEPT ☒ | ALERT ☘ | GATEWAYS GW |
| 5 | ANY ☉ | INTERNAL FINANCE ☉ | ANY ☉ | REJECT STOP | MAIL ✉ | DST ⊕ |

RULE BASE SAVED TO '/FW/USERS/MARLUS/CORPORATE.W'

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, 4, and 10, the combination of Kerr with Checkpoint and Shwed clearly makes up for any such perceived deficiency.

17. Kerr in View of Dietz Renders Obvious Claims 1, 4, and 10 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. (Exhibit 22, “Dietz”) resulted from a patent application filed on June 30, 1999.

During prosecution of the '857 patent, Dietz was disclosed by the patentee alongside approximately 20 other references in its Information Disclosure Statement of January 29, 2010, several weeks after the Examiner had declared that the pertinent rejections over the prior art had been overcome. Ex. 40-D (12/11/2009 Final Rejection) at 3 (“Claims 6, 8, 9, 22-24 and 26-28 would be allowable if a terminal disclaimer is filed to overcome the obviousness-type double patenting rejection.”); Ex. 40-F (1/29/2010 IDS). 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). Dietz was never discussed or applied by the Examiner at any time during prosecution of the '857 patent, and no evidence exists that the Examiner considered any of the technical teachings of Dietz to a greater degree than documents are generally considered during a search of PTO file records. *See* MPEP § 2640. Moreover, as combined with Kerr (which was not considered by the Examiner), Dietz is here considered in a new light.

If certain aspects recited in claims 1, 4, and 10 of the '857 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 in light of the background knowledge of one of ordinary skill in the art, 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:

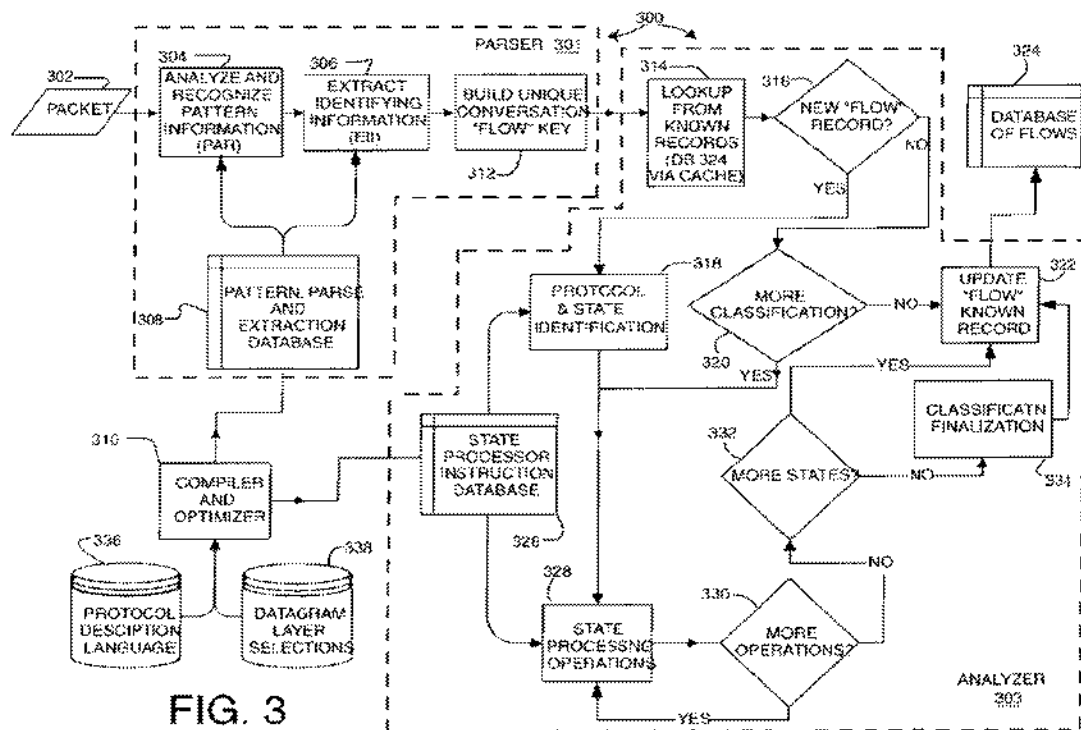


FIG. 3

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, 4, and 10, the combination of Kerr with Dietz clearly makes up for any such perceived deficiency.

18. Kerr in view of Pfeifer96 Renders Obvious Claims 1, 4, and 10 Under § 103

If certain aspects recited in claims 1, 4, and 10 of the '857 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 in light of the background knowledge of one of ordinary skill in the art, under 35 U.S.C. § 103.

(a) Claim 1

i. "A method . . . for processing packets of a message"

Claim 1 recites: "A method in a computer system for processing packets of a message, the method comprising" Under Implicit's apparent claim constructions, Kerr discloses this element. *See* Section V.D.1 (Kerr 102) at Claim 1(i) above.

ii. "receiving a packet of the message"

Claim 1 further recites: "receiving a packet of the message and a data type of the message." Under Implicit's apparent claim constructions, Kerr discloses this element. *See* Section V.D.1 (Kerr 102) at Claim 1(ii) above.

iii. "dynamically identify a sequence of components"

Claim 1 further recites: "analyzing the data type of a first packet of the message to dynamically identify a sequence of components for processing a plurality of packets of the message such that the output format of the components of the sequence match the input format of the next component in the sequence, wherein analyzing the data type of the first packet of the message to dynamically identify the sequence of components includes selecting individual components to form the sequence of components after the first packet of the message is received." Under Implicit's apparent claim constructions, Kerr in view of Pfeifer96 renders obvious this element.

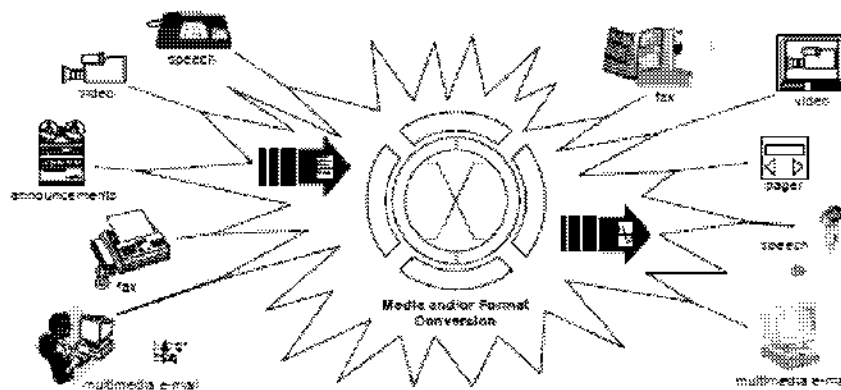
The essential motivation for the Pfeifer96 system is that users are “mobile” and “nomadic,” and still wish to receive their electronic communications even when away from their “well equipped” offices. Ex. 3 at 122, 104. For example, a user may be visiting a location where there is only a telephone, so if a fax is sent to his office he will miss it. *See id.* at 118, 120, 111. Pfeifer96 proposes a general solution to this problem, which relies on three related technologies. First, the system keeps track of the user’s location: users may “register” when they arrive at a location, or may wear an “Active Badge” which tracks their location automatically. *Id.* at 126-27, 119. Second, the system keeps track of the specific “access devices” in the user’s location which would be available to receive incoming communications. *Id.* at 120. Third and most important, if there is not an access device in current location which would be capable of receiving the communication in its original form, the system can dynamically convert the communication into an *entirely different medium*. *Id.* at 120, 124. For example, if there is phone but not a fax machine in the user’s current location, the system can automatically call 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 111, 123-24.

The iPCSS performs this function by observing incoming communications as they enter its “Service Gateways.” *Id.* at 120-24. By doing so, the system can ascertain: (1) the intended destination of an incoming communication (*i.e.*, the called party); and (2) the medium of the incoming communication (*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 communication in its original form. *Id.* at 120. 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 120-24.

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 105, 117 (emphasis in original).

It was obvious to apply this iPCSS system to Kerr, so Kerr could also assure delivery of communications to users in their *actual* locations. As a router, Kerr is well-positioned to observe communications directed to many different users. *E.g.*, Ex. 15 at Figure 1. And as a router, Kerr is well-equipped to convert and re-route these communications as needed for successful delivery.

Kerr already tracks the various pieces of information which would be needed to apply the Pfeifer96 technique. For each incoming flow, Kerr tracks the flow’s “particular destination device” (so Pfeifer96 can determine if the flow would be delivered uselessly to a vacant office), and its medium, *e.g.*, “an internet telephone protocol, or an internet video protocol such as the “CUSeeMe” protocol” (so Pfeifer96 can determine if conversion to another medium is needed to deliver the flow to a device in the user’s vicinity). Ex. 15 (Kerr) at 2:64-65, 3:57-67, 3:13-14.



Ex. 3 (Pfeifer96) at 118 (showing conversion of incoming telephone and video data)

When the first packet of a new 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 the

packets of the flow should be routed next. *Id.* at 1:33-35, 4:13-67. It is clearly at this moment, when determining the proper processing for the flow, that Kerr would simply query the Pfeifer96 system to determine if conversion and re-routing of the flow would be appropriate. *I.g.*, if Pfeifer96 determines the user is away from 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. 3 (Pfeifer96) at 120-24.

Thus, as explained above for Pfeifer96 alone, the combined system would “dynamically identify a sequence of components for processing a plurality of packets of the message.” *See* Section V.C.1 (Pfeifer96 102) at Claim 1(iii) above. The identification is performed by “analyzing the data type of the first packet,” *e.g.*, by determining the medium of the first packet (*e.g.*, “an internet telephone protocol, or an internet video protocol”). Ex. 15 (Kerr) at 3:13-14.

Because Pfeifer96 would dynamically generate the chain in its usual manner, the other aspects of this claim element have been covered above in a section dealing specifically with Pfeifer96. *See* Section V.C.1 (Pfeifer96 102) at Claim 1(iii) above. Additionally, Kerr teaches components which might also be added to a particular sequence. *See* Section V.D.2 (Kerr 103) at Claim 1(iii) above.

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

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

When Kerr has determined the “proper treatment” for a new flow by examining its first packet, it “builds a new entry in the flow cache” and “enters information regarding such proper treatment in a data structure pointed to by the new entry.” Ex. 15 at 4:12-17. “Thus . . . 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 flow cache and treats the packet 150 according to the information in the entry in the flow cache.” *Id.* at 4:64-5:4.

Because the “proper treatment” for packets of the flow would include the converter chain specifically generated for the flow by Pfeifer96, the chain would obviously be stored in the same “data structure,” and for the same reason. *Id.* at 4:15-16, 4:65-66. And thus, Pfeifer96 in view of Kerr renders obvious storing an indication of both the dynamically generated component sequence of Pfeifer96 as well as any indigenous components of Kerr, so the sequence does not need to be re-identified for subsequent packets of the message (flow).

v. “state information”

Claim 1 finally recites: “for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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 Pfeifer96 renders obvious this element.

Because Pfeifer96 in context of Kerr would employ substantially the same sequences of converter components as Pfeifer96 alone, this element has been covered above in a section dealing with Pfeifer96. *See* Section V.C.2 (Pfeifer96 103) at Claim 1(v) above. Additionally, Kerr teaches components which might also be added to a particular sequence. *See* Section V.D.2 (Kerr 103) at Claim 1(v) above.

(b) Claim 4

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

Claim 4 recites: “A method in a computer system for processing a message, the message having a plurality of headers, the method comprising” Under Implicit’s apparent claim constructions, Kerr discloses this element.

As packets arrive at Kerr, they are classified into flows based on the following packet header fields: “IP address for the source device”; “IP address for the destination device”; “protocol type”; “port number for the source device”; and “port number for the destination device.” Ex. 15, at 3:55-65. The first three fields are found in an IP packet’s layer 3 header, and the final two in its layer 4 header (*e.g.*, TCP or UDP).⁴⁶

ii. “dynamically identify a sequence”

Claim 4 further recites: “analyzing the plurality of headers of a first packet of the message to dynamically identify a sequence of components for processing a plurality of packets of the message such that the output format of the components of the sequence match the input format of the next component in the sequence, wherein analyzing the plurality of headers of the first packet of the message to dynamically identify the sequence of components includes selecting individual components to form the sequence of components after the first packet of the message is received.” Under Implicit’s apparent claim constructions, Kerr in view of Pfeifer96 renders obvious this element.

Regarding the limitation “analyzing the plurality of headers of a first packet of the message,” as explained above, Kerr would supply two pieces of information to Pfeifer96 in order

⁴⁶ See, *e.g.*, Ex. 41 (RFC 791) (IP Specification) (1981) at 11 (“Source Address”; “Destination Address”; “Protocol”); Ex. 9 (RFC 793) (TCP Specification) (1981) at 15 (“Source Port”; “Destination Port”). These references are cited in this context solely to help explain Kerr. See MPEP § 2205.

for Pfeifer96 to dynamically generate an appropriate converter chain: (1) the flow's "particular destination device" (so Pfeifer96 can determine whether the flow would be delivered uselessly to a vacant office); and (2) the flow's medium, *e.g.*, "an internet telephone protocol, or an internet video protocol such as the "CUSeeMe" protocol" (so Pfeifer96 can determine if conversion to another medium is needed to deliver the flow to a device in the user's vicinity). *See* Claim 1(iii) above; Ex. 15 (Kerr) at 2:64-65, 3:57-67, 3:13-14. The first is gotten from "the IP address for the destination device" in the first packet's layer 3 header, and the second from a "port number" in its layer 4 header. Ex. 15 at 3:60-61, 3:12-14. Other aspects of this element are discussed above. *See* Claim 1(iii) above.

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

Claim 4 further recites: "storing an indication of each of the identified components so that the sequence does not need to be re-identified for subsequent packets of the message." Under Implicit's apparent claim constructions, Kerr in view of Pfeifer96 renders obvious this element. *See* Claim 1(iv) above.

iv. "state information"

Claim 4 finally recites: "for each of a plurality of components in the identified sequence: performing the processing of each packet by the identified component; 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 Pfeifer96 renders this element. *See* Claim 1(v) above.

(c) Claim 10

Kerr in view of Pfeifer96 renders obvious claim 10. *See* Claim 1 above.

19. Kerr in view of Pfeifer96, ISDN98, and Nelson Renders Obvious Claims 1, 4, and 10 Under § 103

Pfeifer96 has already been combined with Kerr, and ISDN98 and Nelson have already been combined with Pfeifer96 in corresponding sections above. *See* Sections V.D.18 (Kerr+Pfeifer96) and V.C.3 (Pfeifer96+ISDN98+Nelson) above. Those sections should be consulted for the detailed manner of applying them to these references. As applied to the previous combination of Kerr in view of Pfeifer96, ISDN98 and Nelson further confirm that “a plurality of components” in a sequence generated by Pfeifer96 would maintain “state information” across packets in the manner recited by claims 1, 4, and 10. *See id.*

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.

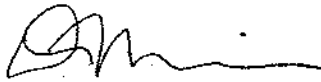
VII. IDENTIFICATION OF REAL PARTY IN INTEREST PURSUANT TO 37 C.F.R. § 1.915(B)(8)

The real party in interest is Juniper Networks, Inc.

VIII. CONCLUSION

The Requester requests that claims 1, 4, and 10 of the '857 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, 4, and 10.

Respectfully submitted,
IRELL & MANELLA LLP



David McPhie, Reg. No. 56,412

Dated: March 2, 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. EM3051485752US and EM305149510US addressed to: Mail Stop *Inter Partes* Reexam, Central Reexamination Unit, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on March 2, 2012.


Susan Langworthy