Transmittal of Communication to Third Party Requester *Inter Partes* **Reexamination**	Control No. 95/000,659	Patent Under Reexamination 6629163	
	Examiner SALMAN AHMED	Art Unit 3992	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address. --

(THIRD PARTY REQUESTER'S CORRESPONDENCE ADDRESS)

IRELL & MANELLA, LLP
DAVID MCPHIE
840 NEWPORT CENTER DR., STE 400
NEWPORT BEACH, CA 92660

Enclosed is a copy of the latest communication from the United States Patent and Trademark Office in the above-identified reexamination prceeding. 37 CFR 1.903.

Prior to the filing of a Notice of Appeal, each time the patent owner responds to this communication, the third party requester of the *inter partes* reexamination may once file written comments within a period of 30 days from the date of service of the patent owner's response. This 30-day time period is statutory (35 U.S.C. 314(b)(2)), and, as such, it cannot be extended. See also 37 CFR 1.947.

If an *ex parte* reexamination has been merged with the *inter partes* reexamination, no responsive submission by any *ex parte* third party requester is permitted.

All correspondence relating to this inter partes reexamination proceeding should be directed to the **Central Reexamination Unit** at the mail, FAX, or hand-carry addresses given at the end of the communication enclosed with this transmittal.

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
95/000,659	02/13/2012	6629163	159291-0025(163)	6219

7590 10/01/2012

HEIM, PAYNE & CHORUSH, LLP
600 TRAVIS STREET
SUITE 6710
HOUSTON, TX 77002

EXAMINER	
AHMED, SALMAN	
ART UNIT	PAPER NUMBER
3992	

MAIL DATE	DELIVERY MODE
10/01/2012	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

PTOL-90A (Rev. 04/07)

Implicit Exhibit 2003
Juniper v. Implicit

ACTION CLOSING PROSECUTION (37 CFR 1.949)

Control No. 95/000,659	Patent Under Reexamination 6629163
Examiner SALMAN AHMED	Art Unit 3992

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address. --

Responsive to the communication(s) filed by:
Patent Owner on 04 June, 2012
Third Party(ies) on 30 August, 2012

Patent owner may once file a submission under 37 CFR 1.951(a) within 1 month(s) from the mailing date of this Office action. Where a submission is filed, third party requester may file responsive comments under 37 CFR 1.951(b) within 30-days (not extendable- 35 U.S.C. § 314(b)(2)) from the date of service of the initial submission on the requester. **Appeal cannot be taken from this action.** Appeal can only be taken from a Right of Appeal Notice under 37 CFR 1.953.

All correspondence relating to this inter partes reexamination proceeding should be directed to the **Central Reexamination Unit** at the mail, FAX, or hand-carry addresses given at the end of this Office action.

PART I. THE FOLLOWING ATTACHMENT(S) ARE PART OF THIS ACTION:

1. ☒ Notice of References Cited by Examiner, PTO-892
2. □ Information Disclosure Citation, PTO/SB/08
3. □ _____

PART II. SUMMARY OF ACTION:

1a. ☒ Claims 1,15 and 35 are subject to reexamination.
1b. □ Claims _____ are not subject to reexamination.
2. □ Claims _____ have been canceled.
3. □ Claims _____ are confirmed. [Unamended patent claims]
4. □ Claims _____ are patentable. [Amended or new claims]
5. ☒ Claims 1,15 and 35 are rejected.
6. □ Claims _____ are objected to.
7. □ The drawings filed on _____ □ are acceptable □ are not acceptable.
8 □ The drawing correction request filed on _____ is: □ approved. □ disapproved.
9 □ Acknowledgment is made of the claim for priority under 35 U.S.C. 119 (a)-(d). The certified copy has:
□ been received. □ not been received. □ been filed in Application/Control No _____
10. □ Other _____

DETAILED ACTION

1. This Office action addresses claims 1, 15 and 35 of United States Patent No. 6,629,163 (Balassanlan, Edward) in response to Patent Owner (hereinafter PO) response dated 6/4/2012 and Third Party Comment dated 8/30/2012 for *inter partes* reexamination.

Information Disclosure Statement

2. The information disclosure statements (IDS) submitted have been considered by the examiner to the extent that they have been explained in the submissions.

Status of the Claims

3. Original claims 1, 15 and 35 are rejected.

Response to Arguments

4. **PO argues in pages 2-3:**

> After Juniper Networks filed its reexamination request, United States District Judge Susan Illston issued a *Markman* Order in the Related Litigation (attached as Exhibit 2, February 29, 2012).[2] The Court's Order—which was not before the PTO when it granted the reexamination request—construed several terms at issue in this reexamination. Although a *Markman* Order is not binding on the PTO, such an Order nonetheless reflects a decision from the District Court on the meaning of particular claim terms in light of the specification and other

intrinsic evidence, and thus should be considered when the PTO construes those same terms for the purpose of reexamination. In this case, Implicit respectfully submits that the PTO should follow the District Court's constructions, in part, because they were rendered after both Implicit and Juniper Networks fully briefed and argued the issues, and because they reflect the considered judgment of an Article III judge. There is no point in re-inventing the (claim construction) wheel in this reexamination.

However, Examiner respectfully disagrees with PO's assertion. "[I]n PTO reexamination, the standard of proof- a preponderance of evidence – is substantially lower than in a civil case" and there is no presumption of validity in reexamination proceedings." *678 F.3d 1357, 1364* (Fed. Cir. 2012) (internal citations omitted); see also *Old Reliable Wholesale, Inc. v. Cornell Corp., 635 F.3d 539, 548 n.6* (Fed. Cir. 2011) ("Whereas clear and convincing evidence is required to invalidate a patent in district court, a patent can be invalidated during PTO reexamination by a simple preponderance of the evidence.").

PO argues in page 11:

The Examiner's positions regarding the proper constructions of the terms in claims 1, 15, and 35 are not clear. For example, while the Office Action includes statements to the effect that "Implicit has taken a broad view" or "under Implicit's apparent claim constructions" (*see, e.g.*, Office Action at 4, 9, 10, and 17), there is no indication how the Examiner believes the claims should be construed. Additionally, the Examiner did not have the benefit of the District Court's claim constructions when the Office Action was issued.

> Patent Owner submits that the PTO should follow the District Court's constructions, which have been fully vetted and argued by both parties. Although the Federal Circuit has held that a *Markman* order in a district court proceeding is not necessarily binding on the PTO, the District Court considered the same claim language and same intrinsic evidence at issue in this reexamination when reaching its constructions. Moreover, after a *Markman* order has issued, the boundaries of the "broadest reasonable interpretation" should reflect the fact that actual boundaries have been identified by a court. *Cf. In re Skvorecz*, 580 F. 3d 1262, 1267 (Fed. Cir. 2009) ("The protocol of giving claims their broadest reasonable interpretation during examination does not include giving claims a legally incorrect interpretation.").

However, Examiner respectfully disagrees with PO's assertion. Examiner has made clear in his office action, his postion by way of rejections, regarding the proper construction of the terms in the claims 1, 15 and 35. Examiner's claim construction in the rejections of the Office Action was based on the Third Party Requester's explaination on claim construciton in the original Request dated 02/13/2012, pages 21-23, incorporated here by reference.

"Claims are given 'their broadest reasonable interpretation, consistent with the specification, in reexamination proceedings.'" *In re Trans Texas Holding Corp., 498 F.3d 1290, 1298* (Fed. Cir. 2007) (quoting *In re Yamamoto, 740 F.2d 1569, 1571* (Fed. Cir. 1984). "Giving claims the broadest reasonable construction 'serves the public interest by reducing the possibility that claims, finally allowed, will be given broader scope than is justified.'" *In re Am. Acad. ofSci. Tech Ctr., 367 F.3d 1359, 1365* (Fed. Cir. 2004) (quoting Yamamoto, 740 F.2d at 1571). "Construing claims broadly during prosecution is not unfair to the applicant (or, in this case, the patentee), because the applicant has the opportunity to amend the claims to obtain more precise claim coverage." *In re Trans Texas Holding Corp., 498 F.3d 1290, 1298* (Fed. Cir. 2007) (citing *Yamamoto, 740 F.2d at 1571-72 and In re Zletz, 893 F.2d 319, 322* (Fed. Cir. 1989)). "The Board is required to use a different standard for construing claims than that used by district courts." *In re Am. Acad. ofSci. Tech Ctr., 367 F.3d 1359, 1369* (Fed. Cir. 2004). Indeed, the Federal Circuit has repeatedly held that "it is error for the Board to 'apply the mode of claim interpretation that is used by courts in litigation, when interpreting the claims of issued patents in connection with determinations of infringement and validity.'" Id. (citing *Zletz, 893 F.2d at 321); see also In re Morris, 127 F.3d 1048, 1054* (Fed. Cir. 1997) ("It would be inconsistent with the role assigned to the PTO in issuing a patent to require it to interpret the claims in the same manner as judges who, post-issuance, operate under the assumption that the patent is valid)." Instead, the PTO is empowered and "obligated to give claims their broadest reasonable interpretation during examination." Id. Thus, in the instant case "the pending claims must be interpreted as broadly as their terms reasonably allow." *In re Zletz, 893 F.2d 321.*

PO argues in pages 12-13:

It is well-settled that the '163 claim terms must be interpreted as they would be by one skilled in the art based on the claim language and in light of the specification. This axiom applies equally to litigation and reexamination. *See Phillips v. AWH Corp.*, 415 F.3d 1303, 1316-17 (Fed. Cir. 2005) (*en banc*) ("The Patent and Trademark Office ('PTO') determines the scope of claims in patent applications not solely on the basis of the claim language, but upon giving claims their broadest reasonable construction 'in light of the specification as it would be interpreted by one of ordinary skill in the art.'"). While the PTO gives claim terms their "broadest reasonable interpretation," it cannot give terms their broadest **possible** interpretation; nor is it permitted to ignore or re-word claim language under the guise of "interpretation."

MPEP § 2111 states: "During patent examination, the pending claims must be 'given their broadest reasonable interpretation **consistent with the specification**.'" *In re Morris*, 127 F.3d 1048, 1054 (Fed. Cir. 1997) (emphasis added). This statement contains two requirements: the interpretation must be "reasonable," and it must be "consistent with the specification." Moreover, the PTO must pay deference to any interpretive guidance offered in the specification, and must view the claims from the perspective of one skilled in the art:

> Some cases state the standard as "the broadest reasonable interpretation," *see, e.g., In re Van Geuns,* (Fed.Cir.1993), others include the qualifier "consistent with the specification" or similar language, *see, e.g., In re Bond,* 910 F.2d 831, 833 (Fed.Cir.1990). Since it would be unreasonable for the PTO to ignore any interpretive guidance afforded by the applicant's written description, either phrasing connotes the same notion: as an initial matter, the PTO applies to the verbiage of the proposed claims the broadest reasonable meaning of the words in their ordinary usage **as they would be understood by one of ordinary skill in the art, taking into account whatever enlightenment by way of definitions or otherwise that may be afforded by the written description contained in the applicant's specification**.

Id. (emphasis added). Thus, the PTO's claim interpretation must follow the meaning accorded by one skilled in this art, when viewed in light of the entirety of the '163 specification. *In re*

Cortright, 165 F.3d 1353, 1359 (Fed. Cir. 1999); *In re Morris*, 127 F.3d at 1054; MPEP § 2111.01-III.

Consistent with these principles, the Federal Circuit recently issued a strong admonishment that the "broadest reasonable interpretation" rule is not a "license" to ignore the words of the claims or the teachings of the specification. In *In re Suitco Surface, Inc.*, 603 F.3d 1255 (Fed. Cir. 2010), the court found that the PTO's construction "though certainly broad, is unreasonably broad." *Id.* at 1260. It stated that "[t]he broadest construction rubric ... does not give the PTO an unfettered license to interpret claims to embrace anything remotely related to the claimed invention. Rather, claims should always be read in light of the specification and teachings in the underlying patent." *Id.* (emphasis added). All of these principles must be followed when construing the '163 claims.

However, Examiner respectfully disagrees with PO's assertion. "[I]n PTO reexamination, the standard of proof- a preponderance of evidence – is substantially lower than in a civil case" and there is no presumption of validity in reexamination proceedings." *678 F.3d 1357, 1364* (Fed. Cir. 2012) (internal citations omitted); see also *Old Reliable Wholesale, Inc. v. Cornell Corp., 635 F.3d 539, 548 n.6* (Fed. Cir. 2011) ("Whereas clear and convincing evidence is required to invalidate a patent in district court, a patent can be invalidated during PTO reexamination by a simple preponderance of the evidence."). "Claims are given 'their broadest reasonable interpretation, consistent with the specification, in reexamination proceedings.'" *In re Trans Texas Holding Corp., 498 F.3d 1290, 1298* (Fed. Cir. 2007) (quoting *In re Yamamoto, 740 F.2d 1569, 1571* (Fed. Cir. 1984). "Giving claims the broadest reasonable construction 'serves the public interest by reducing the possibility that claims, finally allowed, will be given broader scope than is justified.'" *In re Am. Acad. ofSci. Tech Ctr., 367 F.3d 1359, 1365* (Fed. Cir. 2004) (quoting Yamamoto, 740 F.2d at 1571). "Construing claims broadly during prosecution is not unfair to the applicant (or, in this case, the patentee), because the applicant has the opportunity to amend the claims to obtain more precise claim coverage." *In re Trans Texas Holding Corp., 498 F.3d 1290, 1298* (Fed. Cir. 2007) (citing *Yamamoto, 740 F.2d at 1571-72 and In re Zletz, 893 F.2d 319, 322* (Fed. Cir. 1989)). "The Board is required to use a different standard for construing claims than that used by district courts." *In re Am. Acad. ofSci. Tech Ctr., 367 F.3d 1359, 1369* (Fed. Cir. 2004). Indeed, the Federal Circuit has repeatedly held that "it is error for the Board to 'apply the mode of claim interpretation that is used by courts in litigation, when interpreting the claims of issued patents in connection with determinations of infringement and validity.'" Id. (citing *Zletz, 893 F.2d at 321); see also In re Morris, 127 F.3d 1048, 1054* (Fed. Cir. 1997) ("It would be inconsistent with the role assigned to the PTO in issuing a patent to require it to interpret the claims in the same manner as judges who, post-issuance, operate under the assumption that the patent is valid)." Instead, the PTO is empowered and "obligated to give claims their broadest reasonable interpretation during examination." Id. Thus, in the instant case "the pending claims must be interpreted as broadly as their terms reasonably allow." *In re Zletz, 893 F.2d 321*.

Within pages 13-17, PO describes various legal standard for anticipation rejection and legal standards for obviousness rejections.

In response, Examiner submits that, a claim is anticipated where "each and every element as set forth in the claim is found, either expressly or inherently described, in a single prior art reference." See MPEP § 2131, quoting *VerdegaalBros. v. Union Oil Co. of California, 814 F.2d 628, 631* (Fed. Cir. 1987). "When the reference is silent about the asserted inherent characteristic, such gap in the reference may be filled with recourse to extrinsic evidence" without affecting the anticipatory nature of the reference. See

MPEP § 2131.01, quoting *Continental Can Co. USA v. Monsanto Co., 948 F.2d 1264, 1268* (Fed. Cir. 1991). "This modest flexibility in the rule.., accommodates situations in which the common knowledge of" the technologists is not recorded in the reference." Id.

Examiner further submits that, Obviousness may be shown by considering "whether two or more pieces of prior art could be combined, or a single piece of prior art could be modified, to produce the claimed invention." *Innogenetics v. Abbott Labs., 512 F.3d 1363, 1373* (Fed. Cir. 2008). "Prior art is not limited just to the references being applied, but includes the understanding of one of ordinary skill in the art." See MPEP § 2141. Indeed, "the rationale to modify or combine the prior art does not have to be expressly stated in the prior art; the rationale may be expressly or impliedly contained in the prior art or it may be reasoned from knowledge generally available to one of ordinary skill in the art, established scientific principles, or legal precedent established by prior case law. See MPEP § 2144.

In fact, "[i]t is not necessary that the prior art suggest the combination to achieve the same advantage or result discovered by applicant." Id. Prior art from "a different field" is combinable if "it is one which, because of the matter with which it deals, logically would have commended itself to an inventor's attention in considering his problem." *In re Icon Health And Fitness, Inc., 496 F.3d 1374, 1378* (Fed. Cir. 2006); see also *KSR Int'l Co. v. Teleflex, Inc., 127 S. Ct. 1727, 1742* (Fed. Cir. 2007) ("familiar items may have obvious uses beyond their primary purposes"). Examiner has duly provided support for its obviousness rejections in accordance with these principles. PO also urges the Examiner to consider and give weight to supposed secondary indicia of non-obviousness, relying on a declaration from Mr. Edward Balassanian. The law is clear that "[e]vidence of commercial success, or other secondary considerations, is only significant if there is a nexus between the claimed invention and the commercial success." *Ormco Corp. v. Align Tech., Inc., 463 F.3d 1299, 1311-12* (Fed. Cir. 2006); see also *Eurand, Inc. v. Mylan Pharms., Inc., 676 F.3d 1063, 1079* (Fed. Cir. 2012). ("[C]ourts must exercise care in assessing proffered evidence of objective considerations, giving such evidence weight only where the objective indicia are attributable to the inventive characteristics of the discovery as claimed in the patent."). As will be shown below, PO has failed entirely to put forth any competent evidence of secondary indicia of non-obviousness.

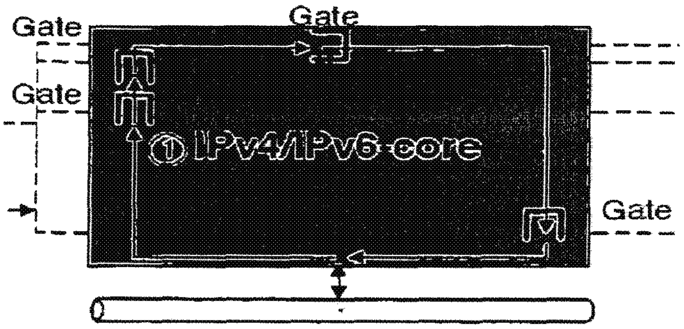
<u>PO argues in page 17:</u>

Decasper describes a router architecture that consists of an IP router core containing a predefined sequence of numerous "components" and "gates" that process each packet received by the router. As explained in the next section, the Decasper router only processes one data format (IP). At the heart of Decasper's router architecture lies an "IPv4/IPv6 core," which "contains the … **components** required for packet processing **which do not come in the form of dynamically loadable modules**." Decasper at 3, col. 2 (emphasis added). Decasper expressly states, therefore, that the IP core "components" are not "dynamically" identified.

Examiner respectfully disagrees with PO's assertion. The Office Action clearly identified Decasper98's **"plugins"** as the claimed **"individual components"** that are dynamically selected to create a sequence in accordance with this claim limitation. The Examiner explained in detail how the process of selection among the various available plugins occurs dynamically, after the first packet is received. OA at 15-17.

PO argues in pages 17-18:

"The core is also responsible for demultiplexing individual packets to plugins" through a series of gates within the IP core. *Id.* When a packet arrives at the router, it is passed to the IP core by the network hardware. *Id.* at 5, col. 2. All packets are processed through the IP core, without exception. Figure 3 of Decasper, which depicts the overall system architecture and data path for an IP router, illustrates the processing path through the IP core via clock-wise arrows:



Excerpt from Figure 3 of Decasper

Each packet proceeds through the IP core in the same manner, until it encounters a gate. "A gate is a point in the IP core where the flow of execution branches off to an instance of a plugin." Decasper at 4, col. 2. Gates are fixed at locations within the routing system where "interactions with plugins need to take place." Decasper at 5, col. 1. Decasper denotes gates in Figure 3 by an "Π" symbol within the IP core. *See* Figure 3 excerpt, above. Plugins are "bound" to a gate during the configuration of the router. Decasper at 4, col 1. The "task" of a gate is to determine if a packet (based only on its header) needs to be processed by a plugin bound to the gate and—if so—which one. Decasper at 5, col 2. These plugins represent extensions to the IP core of the router that provide optional functions such as IP security and packet scheduling. Decasper at 6, col 2. As shown, the location of the gates within the IP core processing path is fixed, and the location of optional plug-in function calls is also fixed (since plug-ins are bound to the gates prior to the receipt of any packets).

Examiner respectfully disagrees with the PO's assertion and agrees with the Third Party Requester's response in pages 7-12. As desribed by the Third Party

Requester, PO virtually ignores discussion of these plugins in its Response, and instead focuses on two other aspects of Decasper98: the "IPv4/IPv6 core" (or "core") and the "gates." PO suggests that certain aspects of the core and gates are not dynamic but rather "fixed," and thus concludes that Decasper98 does not meet the "dynamically identifying" limitation. But even if it were true that all flows traversed the same path through the core and the same sequence of gates (which, as shown below, is incorrect), that fact would have no bearing upon the dynamic identification of plugins in Decasper98, which was the basis for the Examiner's findings regarding this limitation. The plugins of Decasper98 are not the same thing as the core or gates. Decasper98 states that the core only "contains the (few) components required for packet processing which do not come in the form of dynamically loadable modules.". These "few" components in the core are "mainly functions that interact with network devices.". By contrast, the plugins cited in the Office Action are described separately from these few core components, and Decasper98 explicitly states that "all plugins come in the form of dynamically loadable kernel modules." Similarly, although dynamic selection of a plugin may happen at a "gate" (described as "a point in the IP core where the flow of execution branches off to an instance of a plugin"), the gate is not the plugin itself.

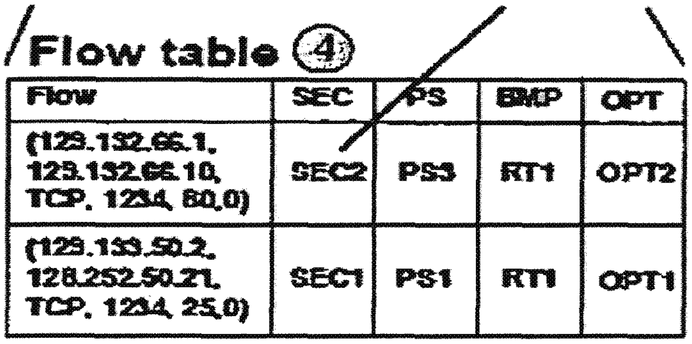
PO argues in pages 18-19:

> It is clear, therefore, that all packets in Decasper receive the same general processing through the IP core, and all packets are processed by the same predetermined number of gates in the same predetermined order. The only difference between the processing paths of packets is

> whether or not they branch out at a specific location (a gate) to a plugin. As the Examiner points out, "The processing of the first packet of a new flow with *n* gates involves *n* filter table lookups." *See, e.g.,* Office Action at 14, 19. Thus—prior to the receipt of the first packet—it has been predetermined that packets will necessarily pass through *n* gates. Furthermore, the sequence of those gates has been predefined.
>
> The above discussion makes clear that the sequence of components and gates within the Decasper IP core is fixed before the first packet arrives. As such, Decasper does not "dynamically identif[y] a non-predefined sequence of components for processing the packets ..., wherein dynamically identifying includes selecting individual components to create the non-predefined sequence of components after the first packet is received."[9]

Examiner respectfully disagrees with the PO's assertion and agrees with the Third Party Requester's response in pages 7-12. As desribed by the Third Party Requester, PO's arguments are clearly directed to the core and gates of Decasper98 and not the plugins. For example, PO argues that "all packets are processed by the same predetermined number of gates in the same predetermined order." Response at 18. PO then jumps seemingly as a *non sequitur* to the conclusion that "[t]he above discussion makes clear that the sequence of components and gates within the Decasper IP core is fixed before the first packet arrives." Response at 19.

However, Decasper98 explicitly discloses an example in which plugin components SEC2, PS3, RT1, and OPT2 are identified for a first flow, and plugin components SEC1, PS1, RT1, OPT1 are identified for a second flow:

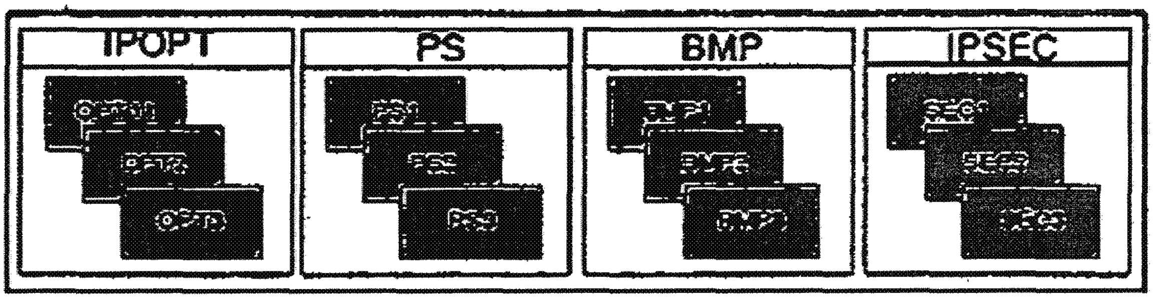


Flow table ④

Flow	SEC	PS	BMP	OPT
(123.132.66.1, 123.132.66.10, TCP, 1234, 80,0)	SEC2	PS3	RT1	OPT2
(123.133.50.2, 128.252.50.21, TCP, 1234, 25,0)	SEC1	PS1	RT1	OPT1

The fact that Decasper98 can be viewed loosely as performing the same general type of processing at a gate (e.g., security processing of some kind at a security gate, packet scheduler processing of some kind at a packet scheduling gate) is irrelevant. It is the selection of individual components (e.g., SEC2 or SEC1 plugin, PS3 or PS 1 plugin) based on characteristics of the first packet received that matters for purposes of the claims.

And indeed, Decasper98's repeated selection from pools of different possible plugins is fundamental to its design as can be seen from the figure below:



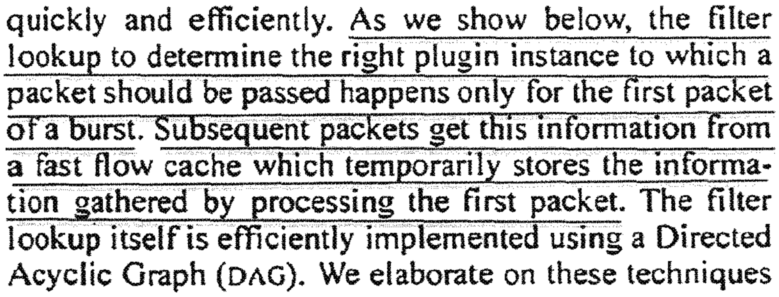
The Decasper98 approach also falls squarely within the scope of the claims as interpreted by PO. For example, PO's Response provides specific examples of protocol stacks as practicing the claims. E.g., Response at page 4 ("OSI model"), 8-9 ("Ethernet" to "IP" to "TCP"; "identifies a sequence" by "examining the packets.., layer by layer").

Thus, PO believes that its invention can cover protocol stacks conforming to the OSI model, even where the same general type of processing is performed at each layer, and in a predetermined order: e.g., a layer 2 component performs data link processing (such as Ethemet), by a layer 3 component performs network processing (such as IP), and a layer 4 component performs transport processing (such as TCP).

However, numerous are the possible sequences of plugins, in fact, that it would generally be "infeasible" to even list them in memory ahead of time. Instead, identification of the sequence of plugins for a flow must wait for the arrival of its first packet, when the various filters can then be applied. Thus, Decasper98 clearly discloses "dynamically identifying a non-predefined sequence of components." Decasper98 also clearly discloses "individual components" that are selected: each individual plugin is selected on the basis of a separate, individual filter table that is matched against the first packet of the flow. Decasper98 also clearly discloses per the claims that this selection is performed "after the first packet is received," since this matching against the first packet cannot take place until after the first packet is received. Accordingly, the plugins of Decasper98 clearly disclose this claim limitation.

Because Decasper98's plugins are dynamically identified and selected as explained above, that claim limitation is satisfied regardless of the characteristics of any other components disclosed in Decasper98. But in any event, PO's argument regarding the core and gates of Decasper98 can also be disregarded on the independent basis that PO's description of these aspects of Decasper98 is technologically incorrect, in at least two ways. First, PO mistakenly alleges that "all packets are processed by the same predetermined number of gates in the same predetermined order" (Response at 18). PO provides no support for this statement, and it is in fact incorrect. Decasper98 does note that the first packet of a particular flow can pass through a sequence of gates, and that a plugin selection will be recorded for each of these gates. But Decasper98 does not state that packets of every flow would pass through that same sequence of gates, and to the contrary, suggests just the opposite. See page 5 ("When a packet arrives" and "makes its way through the core, it may encounter multiple gates"), 7 ("since there is one filter table for every gate in our system, usually multiple lookups (in different filter tables) are necessary for each packet that is received on an uncached flow"). For example, Decasper98 discloses IP security processing in context of a "virtual private network [VPN]." Page 5; Those of ordinary skill understood that for a router to provide a VPN, the VPN packets it receives from a protected network inside must traverse a fundamentally different path than the VPN packets it receives from an unprotected network on the outside. E.g., VPN packets from the protected segment can be encrypted and encapsulated for VPN tunneling only after any operations that require the router to examine or alter the original packet's contents (such as processing its IP options). Conversely, VPN packets from the outside must be unencapsulated and decrypted before any operations that require the router to examine the previously encapsulated packet's contents (such as processing its IP options). One of ordinary skill would appreciate these two distinct and largely opposite operations simply could not take an identical path through the core, and Decasper98 never states they would. A VPN flow from the inside would necessarily traverse a first path through the core and be assigned a first set of components in a first order, while a VPN flow from the outside would traverse a second path and be assigned a second set of components in a second (and largely opposite) order. There is no Other apparent way that the system of Decasper98 could function as described (e.g., to provide a VPN). Thus, PO is incorrect when it states packets are processed at a predetermined number of gates in a predetermined order--as confirmed by language in Decasper98 itself. Ex. at page 5 ("can"; "may") 7 ("usually").

Second, PO appears to suggest a plugin must be selected for every gate that can be encountered by the packets of a particular flow. See, e.g., Response at pages 18-19 ("It is clear, therefore, that all packets in Decasper receive the same general processing through the IP core"). Again, assuming this is PO's suggestion, it is reading something into Decasper98 that is not there. For each gate, there is a set of possible plugins, each with its own filter for matching against the flow's first packet. At page 5. The set of these filters comprises the filter table for a gate. If multiple filters match at a gate, the best ("most specific") match is selected. Pages, 5, 7-8. However, it is certainly possible that no filter at the gate would match, in which case no plugin would or could be selected for the gate. E.g., at page 7 (Table 1) (showing a filter table for a gate which could assign a plugin only to flows having source address "129.* or "128.252.153.*"). For example, Decasper98 explains: "sometimes after a packet is received by the hardware, IP security processing has to be done if the system is configured as an entry point into a virtual private network."). And indeed, it was well understood by those of ordinary skill in the art that "sometimes" a router must perform security processing (e.g., on flows to other branches of the company within the corporate VPN), but "sometimes" it must not (e.g., on flows to public websites such as Yahoo or Google, outside the corporate VPN). And, of course, Decasper98's filter tables would be configured accordingly. Thus, Decasper98 could assign components SEC2, PS3, RT1, OPT2 to a first flow and merely PSI, RT1 to a second (skipping any security and IP options processing entirely. Thus, to the extent PO bases its argument on the notion that packets in Decasper98 all follow the same predefined path through the core or all traverse the same predefined sequence of gates, that argument fails as relying on an incorrect technological reading of Decasper98. In summary, Decasper98 clearly teaches that, after receiving a first packet, the system dynamically selects and identifies individuals plugins to create a sequence that is stored in the flow table for future packets in the same flow.



quickly and efficiently. As we show below, the filter lookup to determine the right plugin instance to which a packet should be passed happens only for the first packet of a burst. Subsequent packets get this information from a fast flow cache which temporarily stores the information gathered by processing the first packet. The filter lookup itself is efficiently implemented using a Directed Acyclic Graph (DAG). We elaborate on these techniques

Accordingly, Decasper98 satisfies this limitation.

PO argues in page 20:

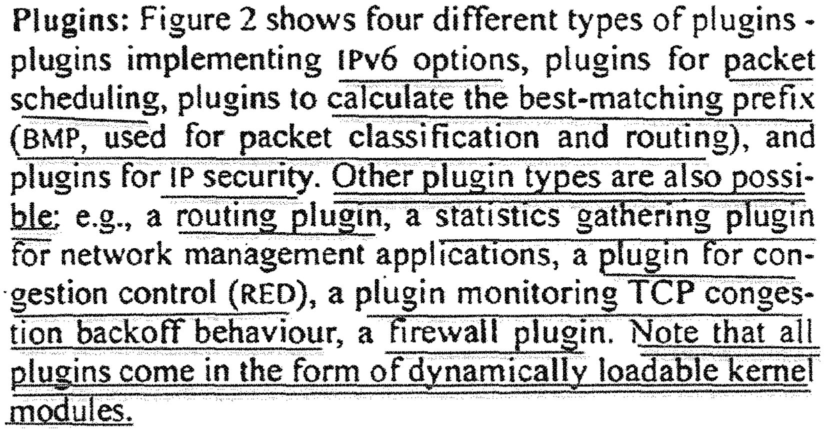
As an initial matter, the obviousness law requires that a prior art reference must be considered in its entirety, including portions that would lead away from the claimed invention. *W.L. Gore & Assoc., Inc. v. Garlock, Inc.*, 721 F.2d 1540, 1552-53 (Fed. Cir. 1983). Decasper discloses an architecture for an IP routing system. An IP router and its components, by definition, are only designed to handle packets in the IP (Internet Protocol) format—a single format. Accordingly, the Decasper architecture assumes that all packets will be in IP format, and therefore that the input and output of every component will be a packet with an IP format. Given the single format of data (IP) that Decasper processes, it teaches away from considering format compatibility when "dynamically identifying a non-predefined sequence of components."

Again, PO's arguments miss the mark as a threshold matter because they fail to address the actual aspects of Decasper98 that the Examiner relied upon in the Office Action. The Office Action clearly identified Decasper98's **"plugins"** as the claimed **"individual components"** that are dynamically selected to create a sequence in accordance with this claim limitation. The Examiner explained in detail above, how the process of selection among the various available plugins occurs dynamically, after the first packet is received. OA at 15-17.

PO argues in page 20:

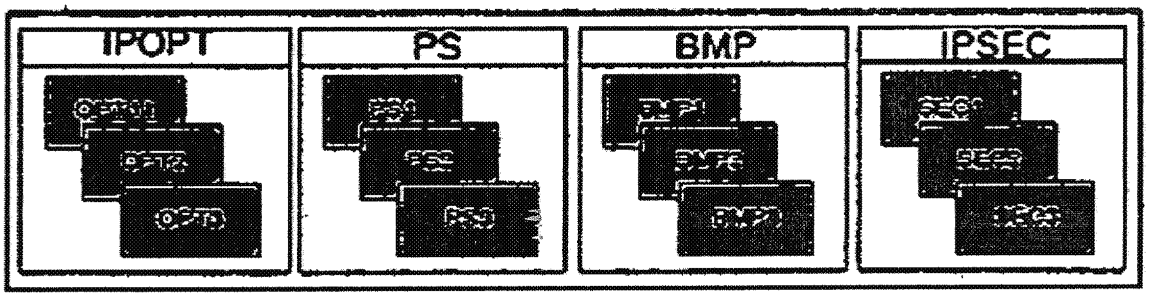
> Nor would there be any "reason"—as required by *KSR*—to consider format compatibility in Decasper. The architecture is specifically designed to handle one format (IP), and thus, one of ordinary skill would be unconcerned about matching different formats. In fact, adding functionality to check or compare the output format of a packet processed by one IP component with the input format required by the next IP component, would add unnecessary and costly overhead. In short, no skilled person looking at Decasper would add the claim requirement.

However, Examiner respectfully disagrees with PO's asseriton. The Office Action clearly stated "component" being "plug-in" and "formats" being various Extended Integrated Service Router (EISR) related messaging of the packet. Decasper98 defines the plug-ins as follows in page 5:



Plugins: Figure 2 shows four different types of plugins - plugins implementing IPv6 options, plugins for packet scheduling, plugins to calculate the best-matching prefix (BMP, used for packet classification and routing), and plugins for IP security. Other plugin types are also possible: e.g., a routing plugin, a statistics gathering plugin for network management applications, a plugin for congestion control (RED), a plugin monitoring TCP congestion backoff behaviour, a firewall plugin. Note that all plugins come in the form of dynamically loadable kernel modules.

Clearly, IP security related format handling related to VPN will be different then that of non-IP security related format handling.



Similarly, IP security related format handling related to VPN will be different then that of Firewall related format handling.

Therefore, Examiner respectfully disagrees with PO's assertion, and submits that within IP, various plugins handle various different fomating needs (such as, scheduling, classifying, security, statistics, congestion, firewall etc.).

PO argues in page 20:

> Not only is there no reason to add format compatibility to Decasper, the Office Action fails to cite to a single piece of evidence for adding such a feature. *See* Office Action at 21. There is no cited patent or printed publication or any other document; instead, there are only bald assertions without the evidentiary support required by the law.

However, Examiner respectfully disagrees with PO's assertion. Examiner stated in prior Office Action page 21:

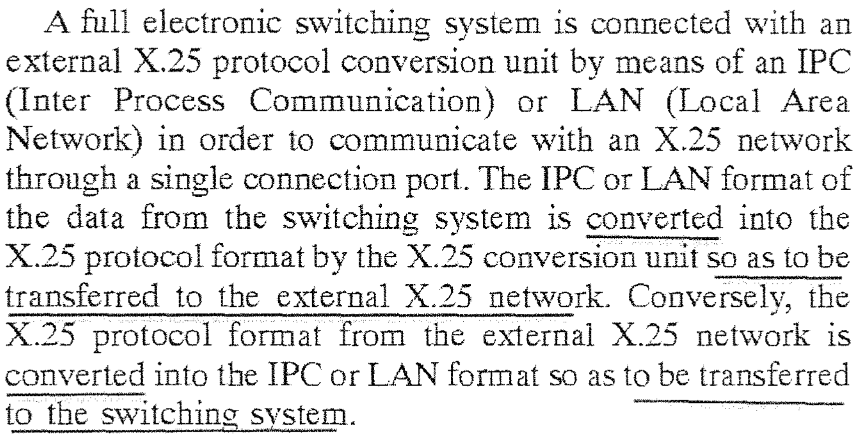
Regarding the limitation "such that the output format of the components ... match the input format of the next component," it was well-known to those of ordinary skill in the art that certain operations on a packet must be performed in a certain order: e.g., if a packet is first converted into an encrypted format by a first component, a subsequent component would be unable to, e.g., process any IPv6 option headers in the packet, or to insert any new ones (because it was expecting to receive the packet in an unencrypted format). Thus, it was certainly obvious for one of ordinary skill in the art to arrange the sequence of components in a compatible manner, such that the output format of one matches the input format of the next.

MPEP 2144.03 Section D states:

If the examiner adds a reference in the next Office action after applicant's rebuttal, and the newly added reference is added only as directly corresponding evidence to support the prior common knowledge finding, and it does not result in a new issue or constitute a new ground of rejection, the Office action may be made final...See MPEP § 706.07(a).

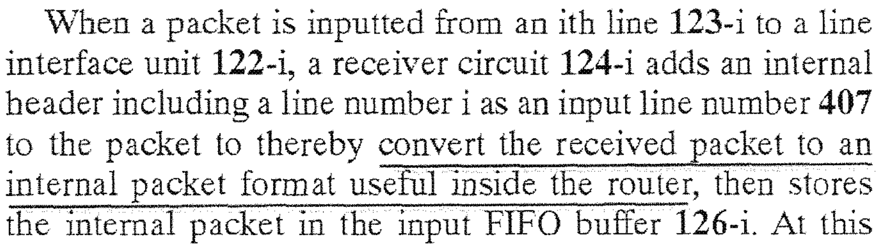
Examiner produces the following as directly corresponding evidence to support the prior common knowledge that output format of the components match the input format of the next component:

US PAT 6192409 states in column 1:



A full electronic switching system is connected with an external X.25 protocol conversion unit by means of an IPC (Inter Process Communication) or LAN (Local Area Network) in order to communicate with an X.25 network through a single connection port. The IPC or LAN format of the data from the switching system is converted into the X.25 protocol format by the X.25 conversion unit so as to be transferred to the external X.25 network. Conversely, the X.25 protocol format from the external X.25 network is converted into the IPC or LAN format so as to be transferred to the switching system.

US PAT 6768738 states in column 13:



When a packet is inputted from an ith line **123**-i to a line interface unit **122**-i, a receiver circuit **124**-i adds an internal header including a line number i as an input line number **407** to the packet to thereby convert the received packet to an internal packet format useful inside the router, then stores the internal packet in the input FIFO buffer **126**-i. At this

Clearly, it was well known, and certainly obvious for one of ordinary skill in the art to arrange the components in a compatible manner, such that the output format of one component matches the input format of the next. Examiner has cited these patents as evidentiary support of the well-known aspect of the knowledge of ordinary skill in the art during the time of the invention.

PO argues in pages 20-21:

Moreover, those assertions are technically inaccurate. As noted above, the Examiner states that "it was well-known to those of ordinary skill in the art that certain operations of 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. process any IPv6 option headers in the pack, or to insert new ones (because it was expecting to receive the packet in unencrypted format)." Office Action at 21. The Examiner's logic seems to be based on a faulty understanding of how IP routers function. First, it should be reiterated that the packets processed by the router all have the same format—that is, they are all IP packets. Second, "encryption" would be performed upon the contents of a packet—not the packet header that Decasper inspects. As a fundamental tenant of IP routing, the packet header contains information that can be rewritten, but is never encrypted, as it must remain a well-formatted IP header in order for other routers to accept it and continue the routing process. Only the content within the IP packet is encrypted, that is, encryption takes place in the transport layer and above, thus preserving the format of the packet as an IP packet. Third, there is no evidence that the plugins in Decasper depend on any input or output constraints other than IP. Even the Examiner's example of encryption does not follow because the gates do not know if the previous plugin did or did not encrypt the packet—they merely assume IP packets. Accordingly—contrary to the assumption of the Examiner—a subsequent component would be able to process the encrypted IP packet, because the packet would still be in IP format and its header would not be encrypted. Thus, the Examiner's argument that it is well-known that encryption of a packet would prevent subsequent components from processing the header of the packet is incorrect.

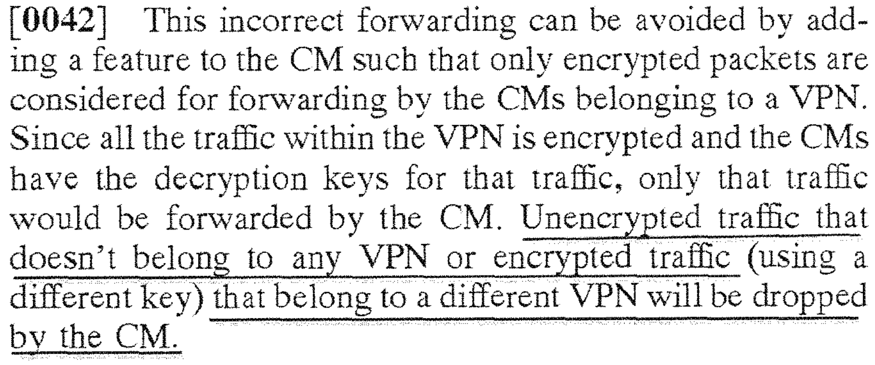
However, Examiner respectfully disagrees with PO's assertion. Clearly, it was well known, and certainly obvious for one of ordinary skill in the art to arrange the components in a compatible manner, such that the output format of one component matches the input format of the next. If a component is expecting encrypted data, it will not process unencrypted data and *vice versa*. Therefore, components have to comply with formatting of packets for successful and error-free processing.

MPEP 2144.03 Section D states:

If the examiner adds a reference in the next Office action after applicant's rebuttal, and the newly added reference is added only as directly corresponding evidence to support the prior common knowledge finding, and it does not result in a new issue or constitute a new ground of rejection, the Office action may be made final...See MPEP § 706.07(a).

Examiner produces the following as directly corresponding evidence to support the prior common knowledge that output format of the components match the input format of the next component:

US PAT PUB 2005/0265309 states in paragraph [0042]:



[0042] This incorrect forwarding can be avoided by adding a feature to the CM such that only encrypted packets are considered for forwarding by the CMs belonging to a VPN. Since all the traffic within the VPN is encrypted and the CMs have the decryption keys for that traffic, only that traffic would be forwarded by the CM. Unencrypted traffic that doesn't belong to any VPN or encrypted traffic (using a different key) that belong to a different VPN will be dropped by the CM.

Examiner has cited the patent publication as evidentiary support of the well-known aspect of the knowledge of ordinary skill in the art.

Examiner, furthermore, agrees with Third Party's response regarding IP routers. Third Party states in pages 14-21 that, the very thing that PO states can "never" happen in an IP router is, in fact, a very common IP routing operation: i.e., the Tunnel-mode encryption which is used to provide virtual private networks (VPNs). It is clear that IP security plugins of Decasper98 would perform this very type of Tunnel-mode encryption to provide VPN functionality. Decasper98 expressly discloses security processing in context of a "virtual private network," and expressly cites to a well-known security standard from 1995 (RFC 1825: "Security Architecture for IP") to explain the operation of its IP security plugin components. At pages 5, 2 ("plugins for IP security" citing footnote "2"), 12 (footnote "2" being "RFC 1825"); IP routers are gateways, and RFC 1825 teaches "Gateway-to-gateway encryption.., for building private virtual networks across an untrusted backbone such as the Internet." Ex. 26 at 4. RFC 1825 explains there are "two modes" for performing encryption:

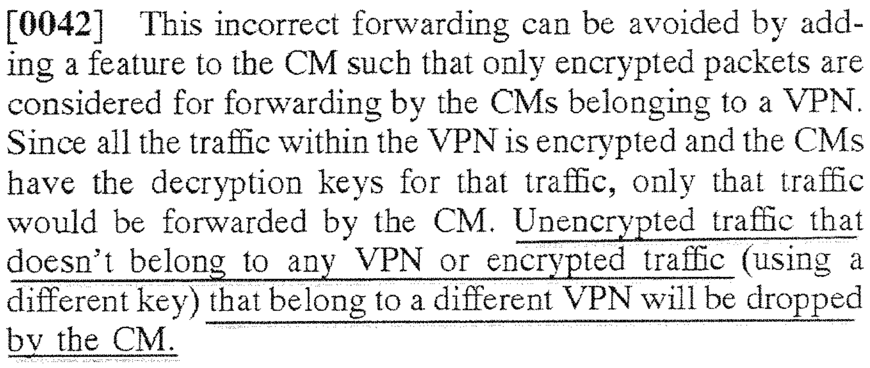
(1) "Tunnel-mode," which encrypts and **"encapsulates an entire IP datagram"** (including its IP header). Ex. 26 at 9. **PO argues this mode is impossible.** Response at 21 ("the packet header.., is never encrypted, as it must remain a well-formatted IP header in order for other routers to accept it and continue the routing process").

(2) "Transport-mode," which encrypts and "encapsulates" only "an upper-layer [transport] protocol (for example UDP or TCP)." Ex. 26 at 9. This is the mode described by PO in its Response. Response at 21 ("encryption takes place in the transport layer and above").

A router operating in Tunnel-mode encrypts the entire original IP packet from a host (including the packet's original destination), and prepends a new, second header to the packet. This second header can be used to address the packet to a second router (as if through a "tunnel") across an untrusted backbone. See Ex. 26 at 4-5, 9. When the second router receives the packet, it decrypts the original packet in its entirety, examines its first, original header, and forwards the packet to its original destination. Two distinct headers (one encrypted, one not) permit delivery to two distinct destinations along the route (first to another router, then to an end host). See Ex. 26 at 4-5, 9. This is the classic VPN design, and those of ordinary skill would certainly understand that Tunnel-mode encryption (as opposed to Transport-mode) would be used by Decasper98's security plugins to provide Decasper98's "virtual private network" feature. See Ex. 25 at 5. In contrast, Transport-mode is typically used for only the special case of "host-to-host encryption," where there is no intervening gateway "present in the connection" that would perform encryption for hosts as a service. See Ex. 26 at 4-5, 9. Thus, it would be unusual for the security plugins of Decasper98 to perform Transport-mode encryption on any of the flows traveling through the Decasper98 router.

Thus, PO's assertion is clearly inaccurate. To provide a VPN, Decasper98's security plugins clearly could and would perform "Tunnel-mode" encryption in which the entirety of the original packet including its header is encrypted. And the type of encryption cited by PO (i.e., Transport-mode) is simply not apposite to the flows through the router that are disclosed by Decasper98.

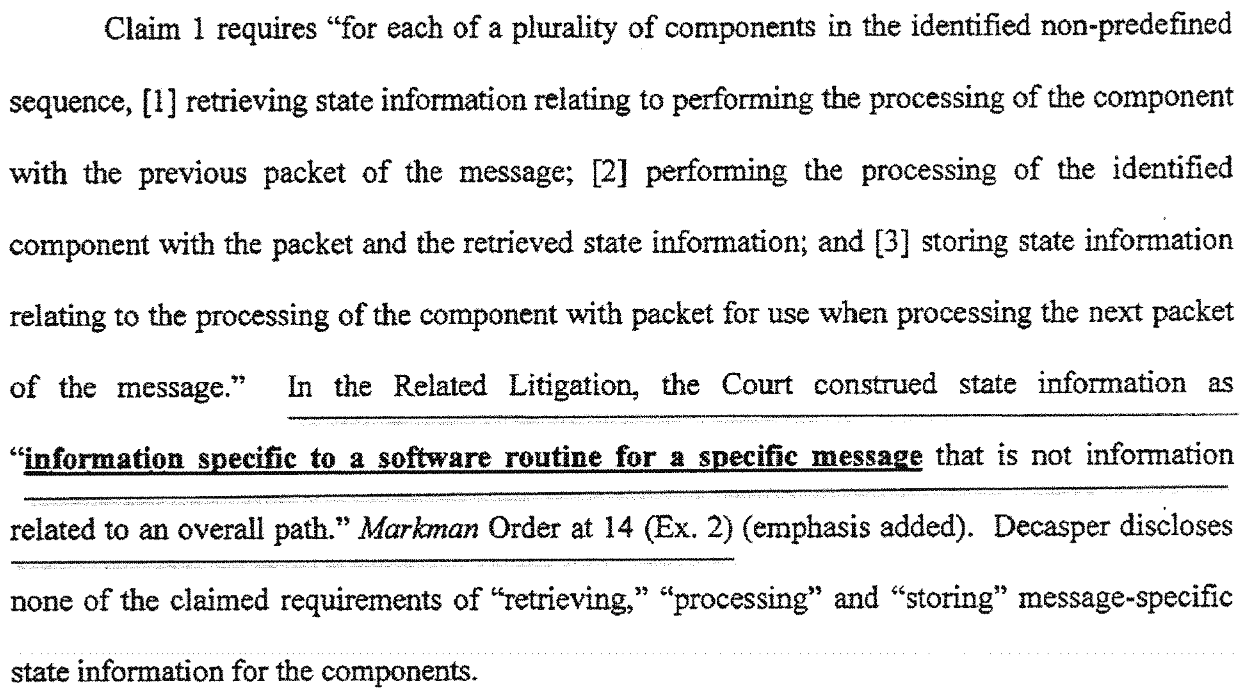
Once a packet has been encrypted for transit through a VPN tunnel, any subsequent components cannot meaningfully read or write any of the packet's original header fields. This format incompatibility (encrypted vs. non-encrypted) necessarily imposes a certain order on Decasper98's component operations. If such an order were not observed, the plugins could not perform their functions. Again this can be clearly seen from **US PAT PUB 2005/0265309** which states in paragraph [0042]:



> **[0042]** This incorrect forwarding can be avoided by adding a feature to the CM such that only encrypted packets are considered for forwarding by the CMs belonging to a VPN. Since all the traffic within the VPN is encrypted and the CMs have the decryption keys for that traffic, only that traffic would be forwarded by the CM. Unencrypted traffic that doesn't belong to any VPN or encrypted traffic (using a different key) that belong to a different VPN will be dropped by the CM.

Examiner has cited the patent publication as evidentiary support of the well-known aspect of the knowledge of ordinary skill in the art as per **MPEP 2144.03 Section D**.

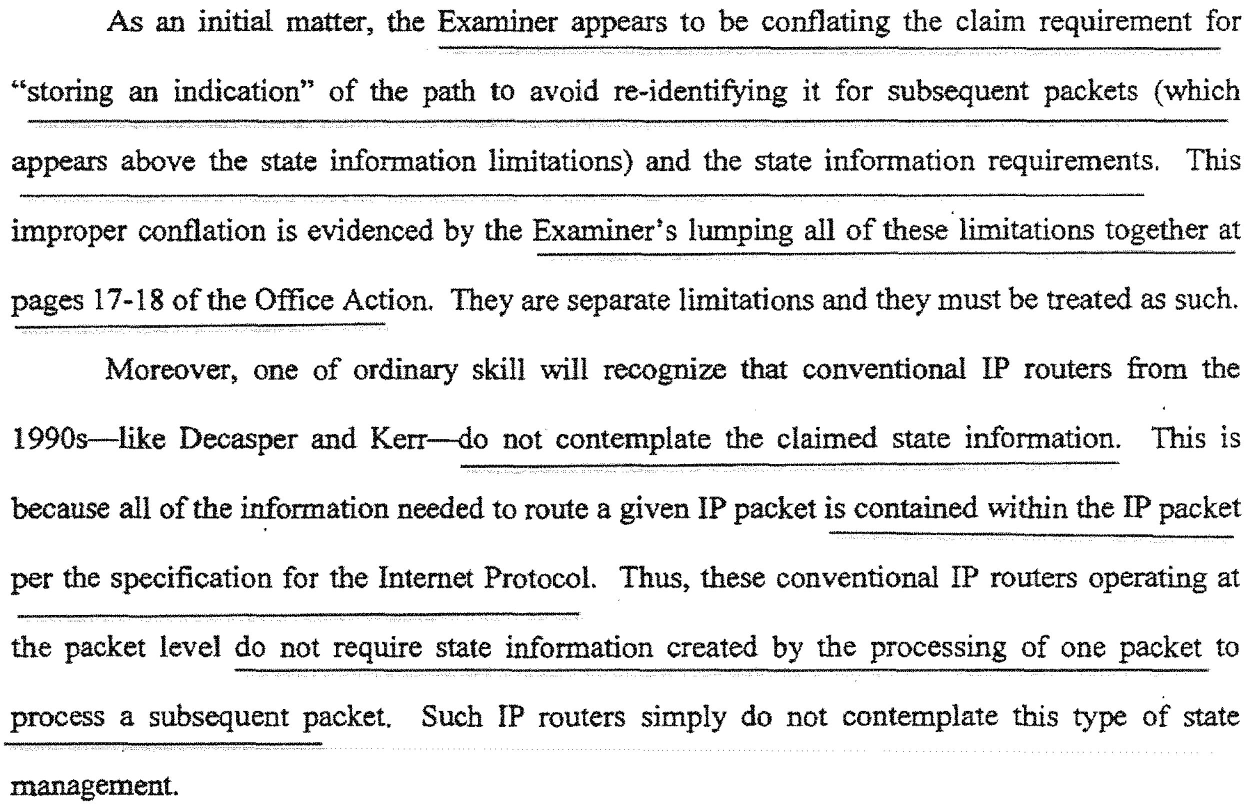
PO argues in page 22:



Claim 1 requires "for each of a plurality of components in the identified non-predefined sequence, [1] retrieving state information relating to performing the processing of the component with the previous packet of the message; [2] performing the processing of the identified component with the packet and the retrieved state information; and [3] storing state information relating to the processing of the component with packet for use when processing the next packet of the message." In the Related Litigation, the Court construed state information as "**information specific to a software routine for a specific message** that is not information related to an overall path." *Markman* Order at 14 (Ex. 2) (emphasis added). Decasper discloses none of the claimed requirements of "retrieving," "processing" and "storing" message-specific state information for the components.

However, Examiner respectfully disagrees with PO's assertion. "[I]n PTO reexamination, the standard of proof- a preponderance of evidence – is substantially lower than in a civil case" and there is no presumption of validity in reexamination proceedings." *678 F.3d 1357, 1364* (Fed. Cir. 2012) (internal citations omitted); see also *Old Reliable Wholesale, Inc. v. Cornell Corp., 635 F.3d 539, 548 n.6* (Fed. Cir. 2011) ("Whereas clear and convincing evidence is required to invalidate a patent in district court, a patent can be invalidated during PTO reexamination by a simple preponderance of the evidence."). "Claims are given 'their broadest reasonable interpretation, consistent with the specification, in reexamination proceedings.'" *In re Trans Texas Holding Corp., 498 F.3d 1290, 1298* (Fed. Cir. 2007) (quoting *In re Yamamoto, 740 F.2d 1569, 1571* (Fed. Cir. 1984). "Giving claims the broadest reasonable construction 'serves the public interest by reducing the possibility that claims, finally allowed, will be given broader scope than is justified.'" *In re Am. Acad. ofSci. Tech Ctr., 367 F.3d 1359, 1365* (Fed. Cir. 2004) (quoting Yamamoto, 740 F.2d at 1571). "Construing claims broadly during prosecution is not unfair to the applicant (or, in this case, the patentee), because the applicant has the opportunity to amend the claims to obtain more precise claim coverage." *In re Trans Texas Holding Corp., 498 F.3d 1290, 1298* (Fed. Cir. 2007) (citing *Yamamoto, 740 F.2d at 1571-72 and In re Zletz, 893 F.2d 319, 322* (Fed. Cir. 1989)). "The Board is required to use a different standard for construing claims than that used by district courts." *In re Am. Acad. ofSci. Tech Ctr., 367 F.3d 1359, 1369* (Fed. Cir. 2004). Indeed, the Federal Circuit has repeatedly held that "it is error for the Board to 'apply the mode of claim interpretation that is used by courts in litigation, when interpreting the claims of issued patents in connection with determinations of infringement and validity.'" Id. (citing *Zletz, 893 F.2d at 321); see also In re Morris, 127 F.3d 1048, 1054* (Fed. Cir. 1997) ("It would be inconsistent with the role assigned to the PTO in issuing a patent to require it to interpret the claims in the same manner as judges who, post-issuance, operate under the assumption that the patent is valid)." Instead, the PTO is empowered and "obligated to give claims their broadest reasonable interpretation during examination." Id. Thus, in the instant case "the pending claims must be interpreted as broadly as their terms reasonably allow." *In re Zletz, 893 F.2d 321.*

PO argues in page 22:



As an initial matter, the Examiner appears to be conflating the claim requirement for "storing an indication" of the path to avoid re-identifying it for subsequent packets (which appears above the state information limitations) and the state information requirements. This improper conflation is evidenced by the Examiner's lumping all of these limitations together at pages 17-18 of the Office Action. They are separate limitations and they must be treated as such.

Moreover, one of ordinary skill will recognize that conventional IP routers from the 1990s—like Decasper and Kerr—do not contemplate the claimed state information. This is because all of the information needed to route a given IP packet is contained within the IP packet per the specification for the Internet Protocol. Thus, these conventional IP routers operating at the packet level do not require state information created by the processing of one packet to process a subsequent packet. Such IP routers simply do not contemplate this type of state management.

However, Examiner respectfully disagrees with PO's assertion. The "state information" interpreted as Decasper98's "pointers" are not within the IP packet as alleged by the PO. Rather, they reside in the "flow table".

PO argues in page 23:

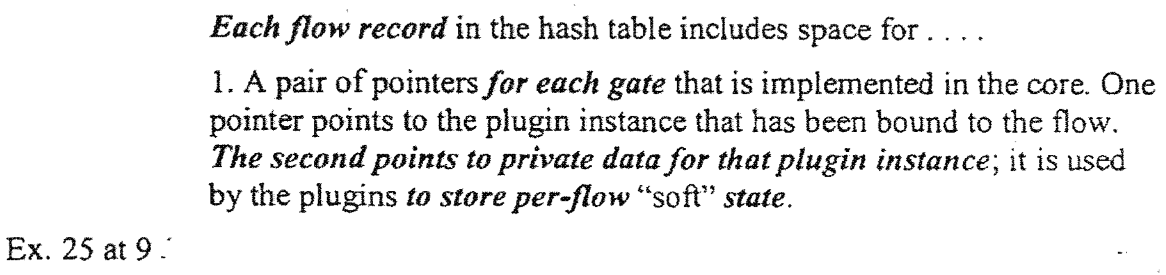
> This accepted wisdom notwithstanding, the Examiner indicates that the "state information" disclosed by Decasper consists of "pointers." Office Action at 18. Although unclear, Patent Owner assumes the Examiner is either referring to the flow index (FIX), which is a "pointer to the row in the flow table where the information associated with the flow is stored," or the "instance pointers" stored within the flow table, which point "to the appropriate plugins for all gates that can be encountered by packets belonging to the corresponding flow," as these are the pointers referenced in the sections of Decasper that the Examiner quoted in the Office Action. Office Action at 19-20. For the reasons discussed below, these pointers do not constitute "information specific to a software routine" that "relate[s] to performing of the processing of the component" with a packet or any other state information limitation, as required by claim 1.

Examiner respectfully states that both pointers, i.e., "flow index" and "instance pointer" are state information. Again regarding "information specific to a software routine" Examiner respectfully submits that "[I]n PTO reexamination, the standard of proof- a preponderance of evidence – is substantially lower than in a civil case" and there is no presumption of validity in reexamination proceedings." *678 F.3d 1357, 1364* (Fed. Cir. 2012) (internal citations omitted); see also *Old Reliable Wholesale, Inc. v. Cornell Corp., 635 F.3d 539, 548 n.6* (Fed. Cir. 2011) ("Whereas clear and convincing evidence is required to invalidate a patent in district court, a patent can be invalidated during PTO reexamination by a simple preponderance of the evidence."). "Claims are given 'their broadest reasonable interpretation, consistent with the specification, in reexamination proceedings.'" *In re Trans Texas Holding Corp., 498 F.3d 1290, 1298* (Fed. Cir. 2007) (quoting *In re Yamamoto, 740 F.2d 1569, 1571* (Fed. Cir. 1984). "Giving claims the broadest reasonable construction 'serves the public interest by reducing the possibility that claims, finally allowed, will be given broader scope than is justified.'" *In re Am. Acad. ofSci. Tech Ctr., 367 F.3d 1359, 1365* (Fed. Cir. 2004) (quoting Yamamoto, 740 F.2d at 1571). "Construing claims broadly during prosecution is not unfair to the applicant (or, in this case, the patentee), because the applicant has the opportunity to amend the claims to obtain more precise claim coverage." *In re Trans Texas Holding Corp., 498 F.3d 1290, 1298* (Fed. Cir. 2007) (citing *Yamamoto, 740 F.2d at 1571-72 and In re Zletz, 893 F.2d 319, 322* (Fed. Cir. 1989)). "The Board is required to use a different standard for construing claims than that used by district courts." *In re Am. Acad. ofSci. Tech Ctr., 367 F.3d 1359, 1369* (Fed. Cir. 2004). Indeed, the Federal Circuit has repeatedly held that "it is error for the Board to 'apply the mode of claim interpretation that is used by courts in litigation, when interpreting the claims of issued patents in connection with determinations of infringement and validity.'" Id. (citing *Zletz, 893 F.2d at 321); see also In re Morris, 127 F.3d 1048, 1054* (Fed. Cir. 1997) ("It would be inconsistent with the role assigned to the PTO in issuing a patent to require it to interpret the claims in the same manner as judges who, post-issuance, operate under the assumption that the patent is valid)." Instead, the PTO is empowered and "obligated to give claims their broadest reasonable interpretation during examination." Id. Thus, in the instant case "the pending claims must be interpreted as broadly as their terms reasonably allow." *In re Zletz, 893 F.2d 321.*

PO argues in page 23:

> Decasper makes it clear that the FIX pointer "is stored in the packet's mbuf" and that "instance pointers cached in the flow table" are retrieved "by accessing the FIX stored in the packet." *Id.* Thus these pointers are not "information specific to a software routine"—the FIX is stored in a packet's buffer, and is therefore the same for each and every plugin. The instance pointers in a flow table are also fixed on a per-packet or per-message basis (rather than a per-component) basis, and point to plug-ins that should process packets of a particular flow. Accordingly, they do not constitute "state information," as this term is used in the claims.

However, Examiner respectfully disagrees with PO's assertion. Examiner has interpreted "state information" to be "pointers" under the broadest reasonable interpretation of the claim language. A proper understanding of Decasper98 clearly reveals plugin-specific functions that maintain message-specific state information across packets. In fact, the designers of Decasper98 deliberately equipped each of its plugin components with a special area for maintaining its own component-specific and message-specific state information across packets. The disclosure of Decasper98 is clear on this point:

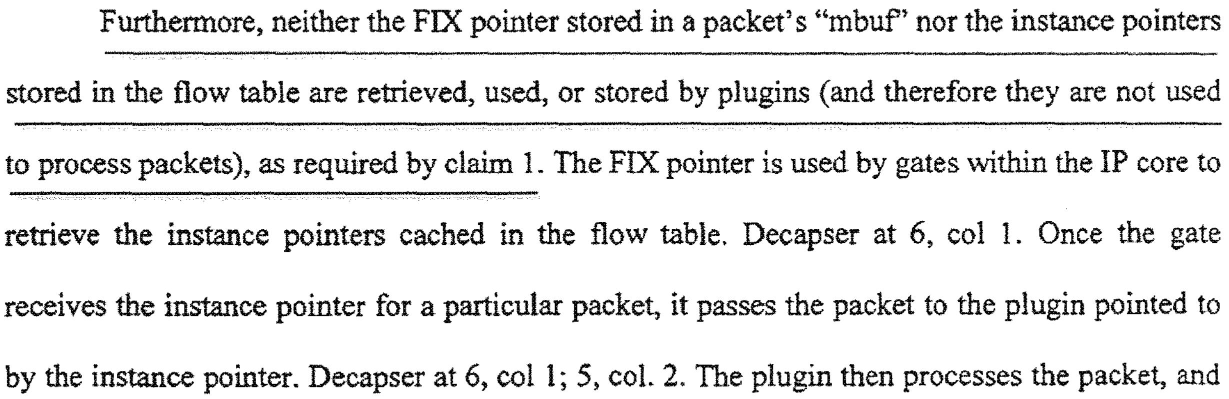


> ***Each flow record*** in the hash table includes space for . . . .
>
> 1. A pair of pointers ***for each gate*** that is implemented in the core. One pointer points to the plugin instance that has been bound to the flow. ***The second points to private data for that plugin instance***; it is used by the plugins ***to store per-flow*** "soft" ***state***.

Ex. 25 at 9.

Therefore, contrarily to PO's assertion, "pointers" store private data for the plugin instance, which is used by the plugins to store per-flow "soft" state.

PO argues in pages 23-24:



Furthermore, neither the FIX pointer stored in a packet's "mbuf" nor the instance pointers stored in the flow table are retrieved, used, or stored by plugins (and therefore they are not used to process packets), as required by claim 1. The FIX pointer is used by gates within the IP core to retrieve the instance pointers cached in the flow table. Decapser at 6, col 1. Once the gate receives the instance pointer for a particular packet, it passes the packet to the plugin pointed to by the instance pointer. Decapser at 6, col 1; 5, col. 2. The plugin then processes the packet, and returns the packet to the IP core. Decasper at 5, col 2. In other words, the sole purpose or function of these pointers is to tell gates where to send a packet for processing—either to a plugin or along the IP core. These pointers are not accessed or utilized by the plugins in any way, and they are certainly not used as the claimed state information. There is no discussion in Decasper about information related to the processing of packets being used by plugins to process the packets, and then stored for use in the processing of a later packet. The plugins of Decapser do not need to know anything about how the previous packet of a message was processed—they simply receive a packet from the IP core, process it, and return it to the IP core. Decasper at 5, Column 2. Accordingly, the use and storage of state information as claimed is not disclosed in Decasper.

However, Examiner respectfully disagrees with PO's assertion. Contrarily to PO's asseriton, the first of the two pointers in the passage cited above comprises information that may be related to the overall path for this flow (recording which plugin was identified for that gate). However, the second of these pointers is not information related to the overall path. It points to a "private data" storage area for use by the individual plugin component only ("private"). Ex. 25 at 9. And Decasper98 explicitly teaches that this private storage area "is used" by the plugin to maintain its own "state" information which is message-specific ("per-flow"). Moreover, Decasper98 automatically provides a multiplicity of these separate state information storage areas, since there is one flow record per flow, and each flow record contains one "private data" area for each of its plugins. For example, if there were 5 flows each with a sequence of 4 plugins, Decasper98 would create fully 20 separate storage areas for the express purpose of storing message-specific state information.

The IP specification was published in 1981 as RFC 79 I, and it does not guarantee that all the information needed by an IP router to route a packet is contained within the packet itself; rather, in some cases it may be necessary for an IP router to maintain message-specific state information across packets. For example, RFC 791 explains "the internet protocol.., provides for fragmentation and reassembly of long datagrams." App. R7 (RFC 791) at 1 is Additionally, IP routers typically perform extensive processing on packets which is not governed by the IP specification, and these operations can require maintaining message-specific state information across packets as well: e.g., gathering network statistics, or applying a stateful encryption algorithm to provide a VPN. See Section V.A.3 (Decasper98 claim 1 "state" limitation). Though PO claims "[t]he '163 system represents a sea-change from the IP routers of Decasper and Kerr" which did "not even contemplate" using "message-specific state information," IP routers did in fact need to maintain message-specific state information across packets simply to perform common router operations. Response at 1.

PO argues in page 24:

Nor do these pointers contain "information relating to the performing of the processing of the component." The instance pointers store the location of the particular plugins to which a gate should send packets associated with a particular flow, and the FIX pointers store the location of the row of the flow table that contains the instance pointers for a particular flow. This "addressing" information does not constitute the claimed "state information."

However, Examiner respectfully disagrees with PO's assertion. Under broadest reasonable interpretation of the claim language, Examiner interprets "pointers" to be "state information". The "pointers" enables per-flow processing of data packets related to plugins. A proper understanding of Decasper98 clearly reveals plugin-specific functions that maintain message-specific state information across packets. In fact, the designers of Decasper98 deliberately equipped each of its plugin components with a special area for maintaining its own component-specific and message-specific state information across packets. The disclosure of Decasper98 is clear on this point:

> ***Each flow record*** in the hash table includes space for . . . .
>
> 1. 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. ***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

Therefore, contrarily to PO's assertion, "pointers" store **private data** for the plugin instance, which is used by the plugins to store per-flow "soft" state. The first of the two pointers in the passage cited above comprises information that may be related to the overall path for this flow (recording which plugin was identified for that gate). However, the second of these pointers is not information related to the overall path. It points to a "private data" storage area for use by the individual plugin component only ("private"). Ex. 25 at 9. And Decasper98 explicitly teaches that this private storage area "is used" by the plugin to maintain its own "state" information which is message-specific ("per-flow"). Id. (emphasis added). Moreover, Decasper98 automatically provides a multiplicity of these separate state information storage areas, since there is one flow record per flow, and each flow record contains one "private data" area for each of its plugins. For example, if there were 5 flows each with a sequence of 4 plugins, Decasper98 would create fully 20 separate storage areas for the express purpose of storing message-specific state information.

If there were not a need to maintain this message-specific, component-specific state information (perflow, perplugin), the Decasper98 design would be needlessly complex and wasteful of memory. For example, instead of 20 distinct state information areas (reflecting state maintained per flow per plugin), the system could be simplified to provide only 5 areas (with state maintained per flow only), or only 4 areas (with state maintained per plugin only), or only 1 area (with global state only). However, it is clear that the designers of the Decasper98 chose as approach that would provide for granularity of state information in a manner consistent with the ' 163 patent claims. Those of ordinary skill in the art would recognize that the plugin functions of Decasper98 would involve maintaining message-specific state information across packets, and would certainly understand that this state information would be maintained in the plugin-specific "private data" area that was provided by Decasper98 for this express purpose.

As a first example of a type of plugin component that would maintain state information in accordance with the '163 patent claims, Decasper98 discloses IP security plugins which perform encryption as defined by RFC 1825 and RFC 1829. Decasper98 cites RFC 1825 which cites RFC 1829, In any event, these RFC's describe a well-known security standard (the "Security Architecture for IP," or "IPsec") that was standard background knowledge for those of ordinary skill reading Decasper98. RFC 1825 explains that an encryption algorithm which "MUST" be supported for IPsec is the "ESP DEC-CBC" algorithm described in RFC 1829. Ex. 26 at 10, 21. RFC 1829 explains that to apply this mandatory encryption algorithm, "an Initialization Vector" must be placed in "[e]ach datagram" to be encrypted (i.e., in each packet)--and that a "common" technique for assigning these vectors "is simply a counter, beginning with a random chosen value." Ex. 27 (RFC 1829) at 1. Thus, an IP security plugin performing such encryption would, for each packet: retrieve the previous counter value; apply it to encrypt the packet; increment the counter value; and store it for use when encrypting the next packet. Additional operations performed by IP security plugins that would entail maintaining state information across packets are cited in the Request. Request at 187-88.

As a second example of a type of plugin component that would maintain message-specific state information across packets, Decasper98 discloses a "statistics gathering plugin." Unless one is reporting statistics about a single packet, it is difficult to see how statistics can be gathered for a flow without maintaining such state information:

e.g., retrieve the previous byte count; add the current packet's length; and store the updated count for use when processing the next packet. See also Request at 188-89.

Additional examples of types of plugin components that would maintain message-specific state information across packets include: IPv6 options plugin components (Request at 188); and packet scheduling plugin components (Request at 189). In short, Decasper98 clearly discloses a storage area that is expressly intended for storing component-specific, message-specific state information, and Decasper98 further discloses a number of plugin components that would use this storage area for that purpose.

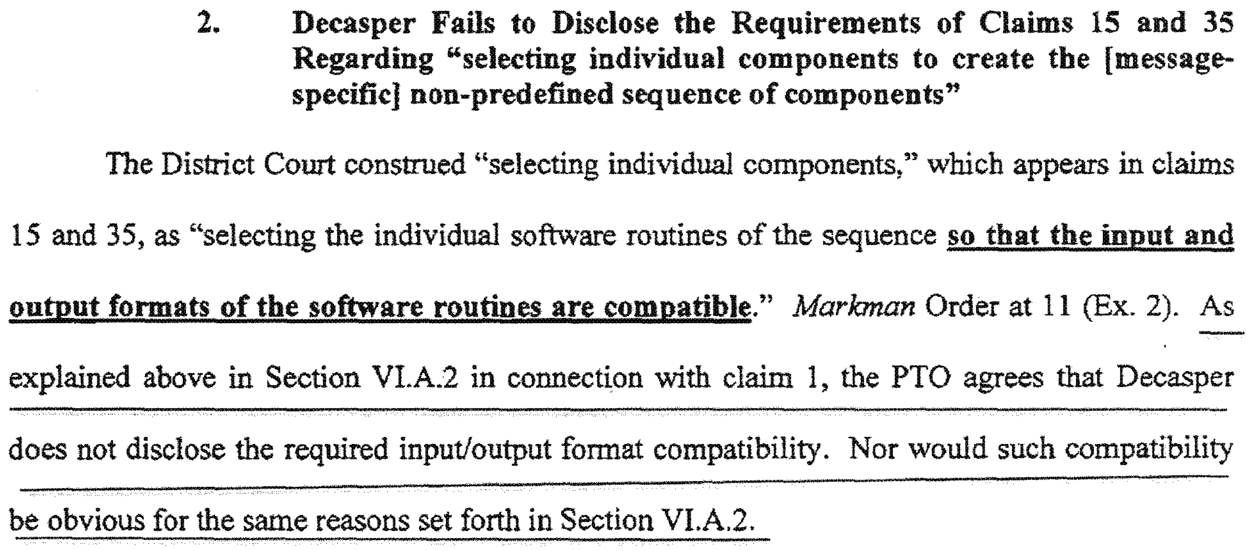
PO argues in page 25:

1. Decasper Fails to Disclose the Requirements in Claims 15 and 35 Regarding Dynamically Identifying a Non-Predefined Sequence of Components

Like claim 1, claims 15 and 35 require "dynamically identifying" a "non-predefined sequence of components." The District Court construed "non-predefined sequence of components" as "a sequence of software routines that was not identified before the first packet of a message was received." *Markman* Order at 6 (Ex. 2). For the same reasons set forth above in Section VI.A.1 with respect to claim 1, Decasper also fails to disclose these requirements in claims 15 and 35.

Examiner respectfully disagrees with PO's assertion. Examiner has shown above that all the cited limitations of claim 1 are taught by Decasper98. Therefore, claims 15 and 35 are not patentable for the same reasons.

PO argues in page 25:

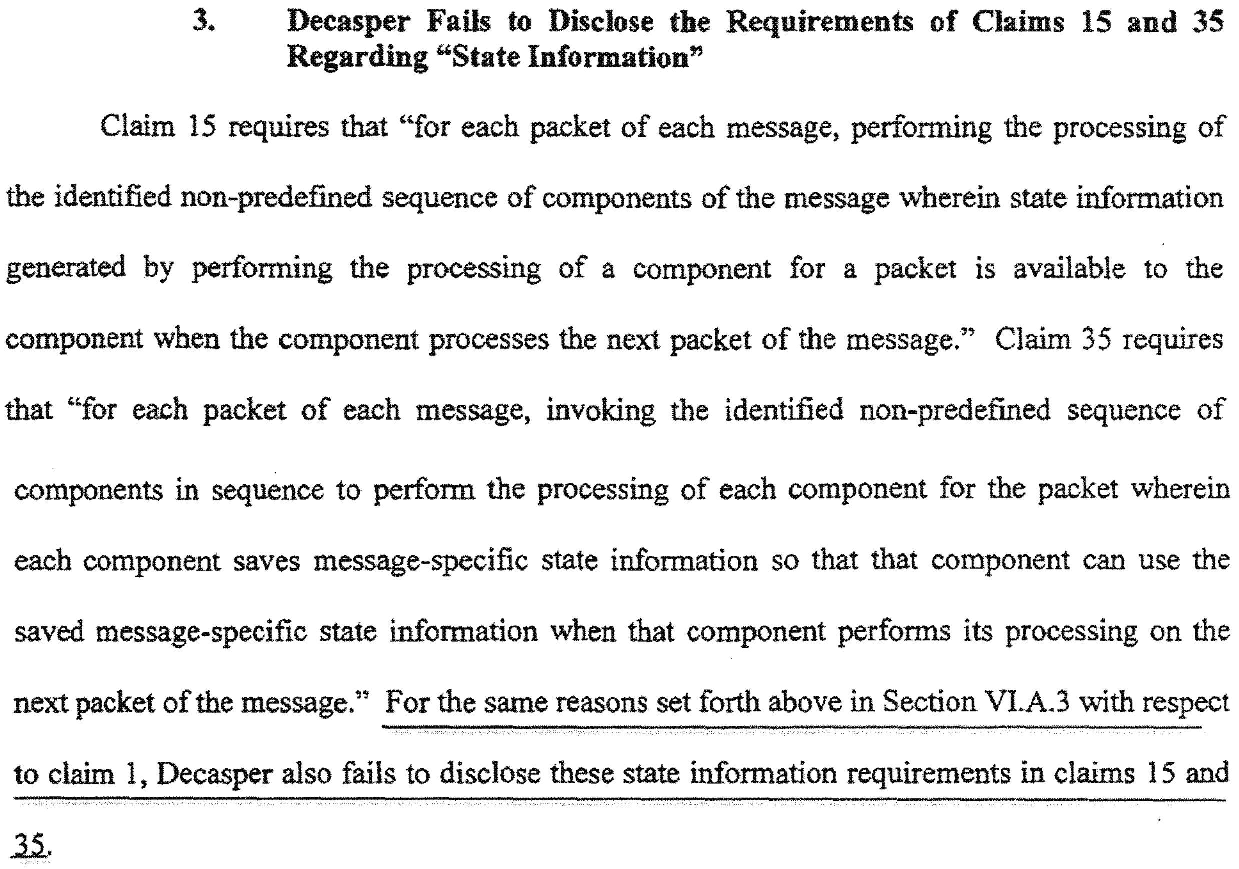


2. Decasper Fails to Disclose the Requirements of Claims 15 and 35 Regarding "selecting individual components to create the [message-specific] non-predefined sequence of components"

The District Court construed "selecting individual components," which appears in claims 15 and 35, as "selecting the individual software routines of the sequence **so that the input and output formats of the software routines are compatible**." *Markman* Order at 11 (Ex. 2). As explained above in Section VI.A.2 in connection with claim 1, the PTO agrees that Decasper does not disclose the required input/output format compatibility. Nor would such compatibility be obvious for the same reasons set forth in Section VI.A.2.

However, Examiner respectfully disagrees with PO's assertion. Examiner has clearly shown above that it was well known, and certainly obvious for one of ordinary skill in the art to arrange the components in a compatible manner, such that the output format of one component matches the input format of the next. Therefore, claims 15 and 35 are not patentable for the same reasons.

PO argues in pages 25-26:



3. Decasper Fails to Disclose the Requirements of Claims 15 and 35 Regarding "State Information"

Claim 15 requires that "for each packet of each message, performing the processing of the identified non-predefined sequence of components of the message wherein state information generated by performing the processing of a component for a packet is available to the component when the component processes the next packet of the message." Claim 35 requires that "for each packet of each message, invoking the identified non-predefined sequence of components in sequence to perform the processing of each component for the packet wherein each component saves message-specific state information so that that component can use the saved message-specific state information when that component performs its processing on the next packet of the message." For the same reasons set forth above in Section VI.A.3 with respect to claim 1, Decasper also fails to disclose these state information requirements in claims 15 and 35.

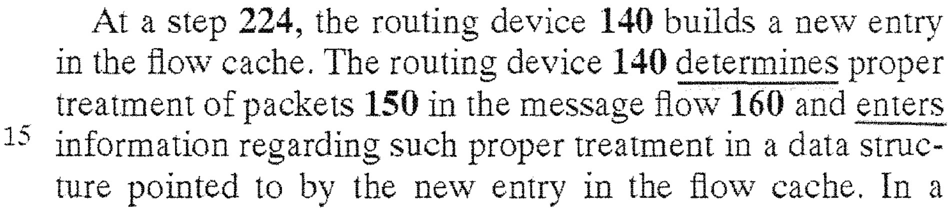
Examiner respectfully disagrees with PO's assertion. Examiner has shown above that all the cited limitations of claim 1 are taught by Decasper98. Therefore, claims 15 and 35 are not patentable for the same reasons.

PO argues in page 27:

> The Examiner points to the language at column 4, lines 20-34 in Kerr as disclosing the dynamic identification of a non-predefined sequence of components. However, this passage does not identify any components (*i.e.*, software routines). Instead, it discusses what the "**proper treatments of packets** … in the message flow" might be, with regards to "switching," "access control," "accounting," or "any special treatment for packets." Importantly, this passage—like all of the other passages in the reference—is discussing the determination of the **treatment** or **processing** of packets. Kerr never discusses or discloses the identification or selection of software routines for performing said processing or treatments, nor the "dynamic identification" of a "non-predefined sequence" of such components. In fact, Kerr does not discuss any sort of software routines.

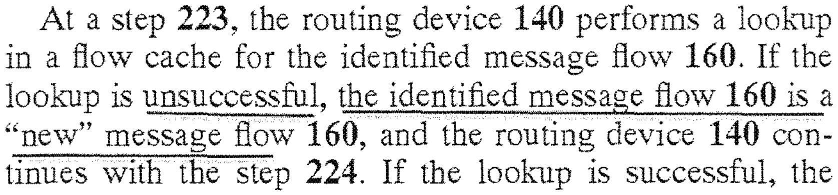
However, Examiner respectfully disagrees with PO's assertion. Under the broadest reasonable interpretation of the claim language, Kerr discloses a "plurality of components" for processing messages. For example, claim 1 of Kerr describes using a "plurality of devices" to apply "policy treatments" to a "plurality of messages," where policy treatments are used to perform "access control, ...security," "queuing," "accounting," "traffic profiling," etc. Id. at 10:27-40. Processing components can include "treatment with regard to switching," "access control," and "encryption." Id. at 4:20-34. "[S]pecial processing" can include "authentication" techniques "useful for implementing security 'firewalls.'" Id. at 35-46. Kerr further discloses that a "rewrite function" may be invoked "to alter the header for the packet." Id. at 4:55-62. These components can be used for "converting data with an input format into data with an output format," for . example, the "encryption" and "rewrite" components to "alter" data to be processed. Id. at 4:30- 31, 4:55-62. The processing components of Kerr comprise "software routine" embodiments, as Kerr states that the processing instrumentality "may include specific hardware constructed or **programmed performing the process steps** described herein" or "a general purpose processor operating under **program control**." Id. at 2:51-55; see also Id. at Figs. 3-4 (illustrating software data structures).

In regards to "identification" of components, Kerr discloses in column 4 lines 12-15:



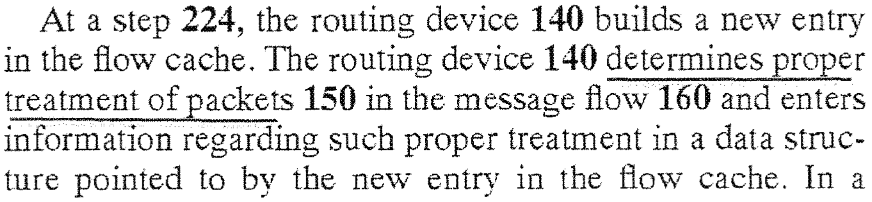
> At a step **224**, the routing device **140** builds a new entry in the flow cache. The routing device **140** determines proper treatment of packets **150** in the message flow **160** and enters information regarding such proper treatment in a data structure pointed to by the new entry in the flow cache. In a

In regards to "dynamic identification" of "non-predefined sequence" of components Kerr discloses in column 4:



> At a step **223**, the routing device **140** performs a lookup in a flow cache for the identified message flow **160**. If the lookup is unsuccessful, the identified message flow **160** is a "new" message flow **160**, and the routing device **140** continues with the step **224**. If the lookup is successful, the

Therefore, routing device is trying to "dynamically identifying" necessary processing components.



> At a step **224**, the routing device **140** builds a new entry in the flow cache. The routing device **140** determines proper treatment of packets **150** in the message flow **160** and enters information regarding such proper treatment in a data structure pointed to by the new entry in the flow cache. In a

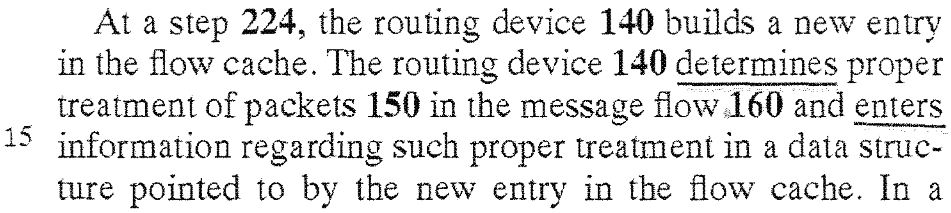
Therefore, Kerr performs "dynamic identification" of "non-predefined sequence" of processing treatments (components) as required by claim language.

PO argues in pages 27 and 28:

Furthermore, Kerr makes clear that it is the information about the complete proper treatment or processing of packets that is stored in the flow cache—not an identification of a particular software component. "The flow cache 300 … includes information about a particular message flow 160, including …access control, accounting, special treatment for packets 150 in that particular message flow, and a pointer to information about treatment of packets 150." Kerr at 6:32-42. When a routing device receives a subsequent packet in a flow, "[it] 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." Kerr at 4:67-5:4.

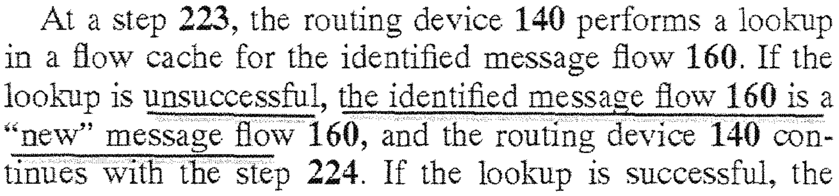
Nothing in the other sections of Kerr that the Examiner cites to (*i.e.*, Kerr at 10:31-40 and 3:38-6:27) discloses anything other than the determination of proper processing and the storage of information about that processing. In fact, the Examiner's own argument repeatedly states that the routing device "determines proper treatment of packets" and "enters information regarding such proper treatment" in a data structure. *See* Office Action at 5, 7.

However, Examiner respectfully disagrees with PO's assertion. As described above, in regards to "identification" of components, Kerr discloses in column 4 lines 12-15:



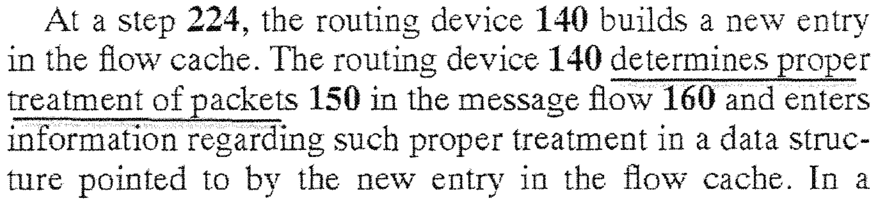
At a step **224**, the routing device **140** builds a new entry in the flow cache. The routing device **140** determines proper treatment of packets **150** in the message flow **160** and enters information regarding such proper treatment in a data structure pointed to by the new entry in the flow cache. In a

In regards to "dynamic identification" of "non-predefined sequence" of components Kerr discloses in column 4:



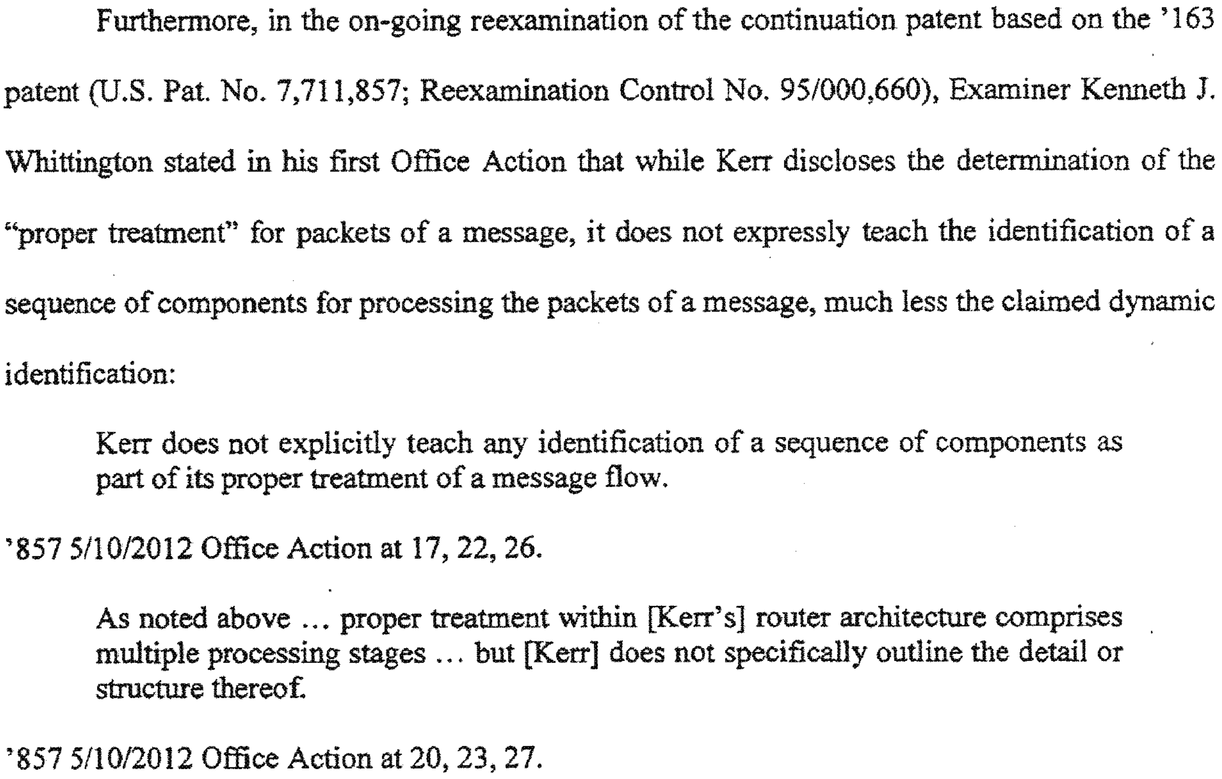
At a step 223, the routing device 140 performs a lookup in a flow cache for the identified message flow 160. If the lookup is unsuccessful, the identified message flow 160 is a "new" message flow 160, and the routing device 140 continues with the step 224. If the lookup is successful, the

Therefore, routing device is trying to "dynamically identifying" necessary processing components.



At a step 224, the routing device 140 builds a new entry in the flow cache. The routing device 140 determines proper treatment of packets 150 in the message flow 160 and enters information regarding such proper treatment in a data structure pointed to by the new entry in the flow cache. In a

PO argues in page 28:



Furthermore, in the on-going reexamination of the continuation patent based on the '163 patent (U.S. Pat. No. 7,711,857; Reexamination Control No. 95/000,660), Examiner Kenneth J. Whittington stated in his first Office Action that while Kerr discloses the determination of the "proper treatment" for packets of a message, it does not expressly teach the identification of a sequence of components for processing the packets of a message, much less the claimed dynamic identification:

> Kerr does not explicitly teach any identification of a sequence of components as part of its proper treatment of a message flow.

'857 5/10/2012 Office Action at 17, 22, 26.

> As noted above … proper treatment within [Kerr's] router architecture comprises multiple processing stages … but [Kerr] does not specifically outline the detail or structure thereof.

'857 5/10/2012 Office Action at 20, 23, 27.

However, Examiner respectfully submits that the scope of the claim limitations are different between the claims of the instant reexamination proceedings and concurrent reexamination proceedings 95/000660. They are spawned from different patents. An example of scope variation of claim limitations is shown below:

Instant Proceeding	95/000660
Claim 1. A method in a computer system for processing a message having a sequence of packets, the method comprising:	Claim 1. A method in a computer system for processing packets of a message, the method comprising:
providing a plurality of components, each component being a software routine for converting data with an input format into data with an output format;	
for the first packet of the message, identifying a 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;	receiving a packet of the message and a data type of the message; 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;
and 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; and for each of a plurality of packets of the message in sequence, for each of a plurality of components in the identified sequence, retrieving state information relating to performing the processing of the component with the previous packet of the message; performing the processing of the identified component with the packet	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; 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.

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

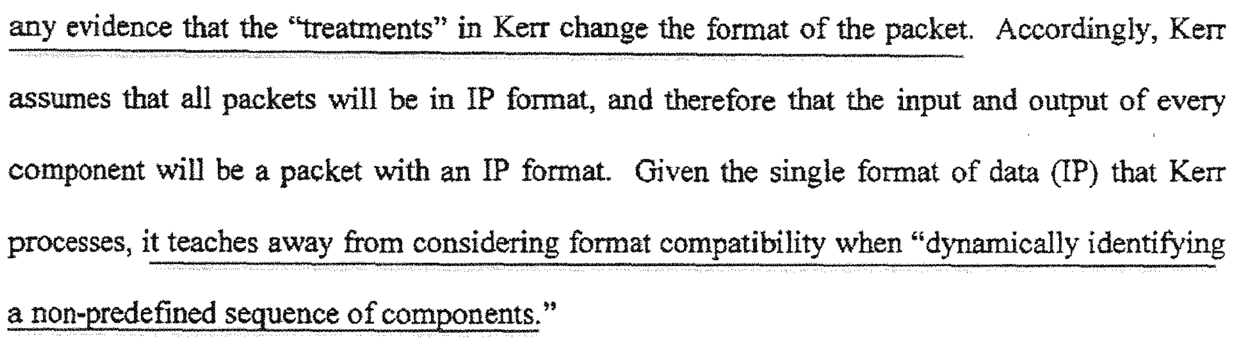
Therefore, based on the scope of the claim limitations, the rejections are articulated for respective proceedings. As such, based on claim interpretation of respective proceedings, different rejections are articulated.

It is for the same reasons, Examiner respectfully disagrees with PO's assertion that Kerr fails to disclose "dynamically identifying a non-predefined sequence of components" for processing the packets of a message and the examiner has also failed to present a *prima facie* case that Kerr anticipates this limitation of claim 1.

PO argues in pages 29-30:

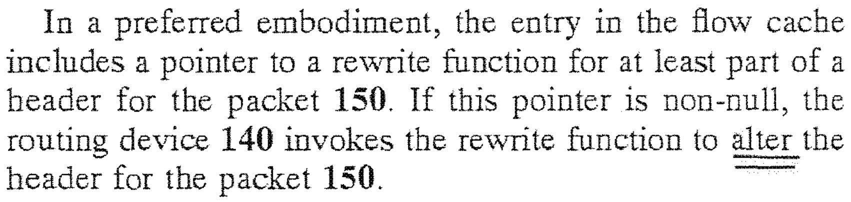
The PTO agrees that Kerr does not disclose "processing the packets of the message such that the output format of the components match the input format of the next component." Office Action at 11. But the examiner states that "it was well-known to those of ordinary skill in the art that certain operations of 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 unencrypted format)." Office Action at 11. The Examiner then summarily concludes that "it was certainly at least obvious for one of ordinary skill in the art to arrange the sequence of components in a compatible manner, such that the output format of one matches the input format of the next—rather than arranging them in an incompatible manner whereby various component(s) would be unable to perform their function(s)." *Id.* The Examiner has failed to establish a *prima facie* case of obviousness because the above statements are both technically incorrect and do not follow the strict legal requirements set forth in Section V.B, above.

As an initial matter, the obviousness law requires that a prior art reference must be considered in its entirety, including portions that would lead away from the claimed invention. *W.L. Gore & Assoc., Inc. v. Garlock, Inc.*, 721 F.2d 1540, 1552-53 (Fed. Cir. 1983). Kerr discloses a system and method for packet handling within an IP router. Kerr never discloses or discusses the format of the data packets processed by its system. Presumably, this is because Kerr limits its disclosure to an IP router and its components, which, by definition, are only designed to handle packets in the IP (Internet Protocol) format—a single format. Nor is there



any evidence that the "treatments" in Kerr change the format of the packet. Accordingly, Kerr assumes that all packets will be in IP format, and therefore that the input and output of every component will be a packet with an IP format. Given the single format of data (IP) that Kerr processes, it teaches away from considering format compatibility when "dynamically identifying a non-predefined sequence of components."

PO's arguments miss the mark as a threshold matter because they fail to address the actual aspects of Kerr that the Examiner relied upon in the Office Action. The Office Action clearly identified Kerr's **"plurality of device"** as the claimed **"individual components"** that are dynamically selected to create a sequence in accordance with this claim limitation. The Examiner explained in detail in the Office Action, how the process of selection among the various available plurality of device occurs dynamically, after the first packet is received. OA at 4-5. Clearly, contrarily to PO's assertion, the format of IP packet clearly changes. One example is in column 4:



> In a preferred embodiment, the entry in the flow cache includes a pointer to a rewrite function for at least part of a header for the packet **150**. If this pointer is non-null, the routing device **140** invokes the rewrite function to alter the header for the packet **150**.

Depending on the flow, its "proper treatment" may include performing various distinct operations such as: "encryption"; header "rewrite"; "switching"; "access control"; "accounting"; and "authenticat[ion]." Kerr at 4:20-60. Therefore, contrarily to PO's assertion, the IP packet format changes depending on "encryption" or header "rewrite" etc.

PO argues in page 30:

> Nor would there be any "reason"—as required by *KSR*—to consider format compatibility in Kerr. Kerr appears to be specifically designed to handle one format (IP), and thus, one of ordinary skill would be unconcerned about matching different formats. In fact, adding functionality to check or compare the output format of a packet processed by one IP component with the input format required by the next IP component, would add unnecessary and costly overhead. In short, no skilled person looking at Kerr would add the claim requirement.
>
> Not only is there no reason to add format compatibility to Kerr, the Office Action fails to cite to a single piece of evidence for adding such a feature. *See* Office Action at 11. There is no cited patent or printed publication or any other document; instead, there are only bald assertions without the evidentiary support required by the law.

As mentioned above, depending on the flow, its "proper treatment" may include performing various distinct operations such as: "encryption"; header "rewrite"; "switching"; "access control"; "accounting"; and "authenticat[ion]." Kerr at 4:20-60. Therefore, contrarily to PO's assertion, the IP packet format changes depending on "encryption" or header "rewrite" etc.

PO argues in page 30:

> Not only is there no reason to add format compatibility to Kerr, the Office Action fails to cite to a single piece of evidence for adding such a feature. *See* Office Action at 11. There is no cited patent or printed publication or any other document; instead, there are only bald assertions without the evidentiary support required by the law.

However, Examiner respectfully disagrees with PO's assertion. Examiner stated in prior Office Action page 21:

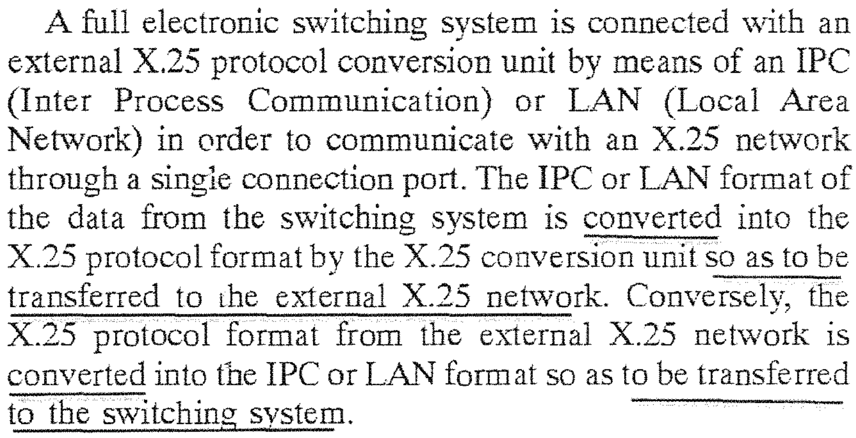
> Regarding the limitation "such that the output format of the components ... match the input format of the next component," it was well-known to those of ordinary skill in the art that certain operations on a packet must be performed in a certain order: e.g., if a packet is first converted into an encrypted format by a first component, a subsequent component would be unable to, e.g., rewrite its headers (because it was expecting to receive the packet in an unencrypted format). See id. at 4:31-32 ("encryption treatment for packets.., in the message flow"), 4:57-58 ("rewrite function for.., a header for the packet"). Thus, it was certainly at least obvious for one of ordinary skill in the art to arrange the sequence of components in a compatible manner, such that the output format of one matches the input format of the next-- rather than arranging them in an incompatible manner whereby various component(s) would be unable to perform their function(s).

MPEP 2144.03 Section D states:

If the examiner adds a reference in the next Office action after applicant's rebuttal, and the newly added reference is added only as directly corresponding evidence to support the prior common knowledge finding, and it does not result in a new issue or constitute a new ground of rejection, the Office action may be made final...See MPEP § 706.07(a).

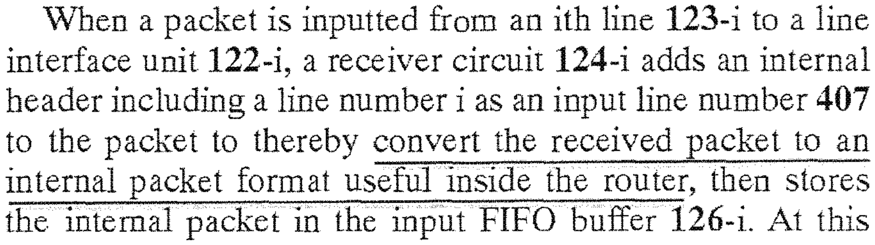
Examiner produces the following as directly corresponding evidence to support the prior common knowledge that output format of the components match the input format of the next component:

US PAT 6192409 states in column 1:



> A full electronic switching system is connected with an external X.25 protocol conversion unit by means of an IPC (Inter Process Communication) or LAN (Local Area Network) in order to communicate with an X.25 network through a single connection port. The IPC or LAN format of the data from the switching system is converted into the X.25 protocol format by the X.25 conversion unit so as to be transferred to ihe external X.25 network. Conversely, the X.25 protocol format from the external X.25 network is converted into the IPC or LAN format so as to be transferred to the switching system.

US PAT 6768738 states in column 13:



> When a packet is inputted from an ith line **123**-i to a line interface unit **122**-i, a receiver circuit **124**-i adds an internal header including a line number i as an input line number **407** to the packet to thereby convert the received packet to an internal packet format useful inside the router, then stores the internal packet in the input FIFO buffer **126**-i. At this

Clearly, it was well known, and certainly obvious for one of ordinary skill in the art to arrange the components in a compatible manner, such that the output format of one component matches the input format of the next. Examiner has cited these patents as evidentiary support of the well-known aspect of the knowledge of ordinary skill in the art during the time of the invention.

PO argues in pages 30-31:

Furthermore, as explained above, Kerr does not discuss any "components" or "sequence of components"—its disclosure is limited to the determination of the proper treatment or processing of packets. Thus, there is also no disclosure of dynamically identifying a non-predefined sequence of components for processing the packets of the message "such that the **output format of the components** of the non-predefined sequence **match the input format of the next component** in the non-predefined sequence."

Finally, the Examiner's assertions are technically inaccurate. As noted above, the Examiner states that the encryption processing disclosed in Kerr could not be performed before a rewrite function for a header of a packet. Office Action at 11. The Examiner's logic is again based on a faulty understanding of how IP routers function, as well as a misunderstanding of the disclosure of Kerr. As explained in Section VI.A.2 above, IP routers are limited to processing IP packets. In addition, encryption is only performed upon the contents within a packet—not the packet's header, while a "header rewrite" function only deals with the header of a packet. As a fundamental tenant of IP routing, the packet header contains information that can be rewritten, but this information is never encrypted—otherwise the packet would not be routable through other routers on the network and the destination device will not accept it. Accordingly—contrary to the assumption of the Examiner—a subsequent component would be able to process the header of an encrypted IP packet, because the packet would still be in IP format and its header would not be encrypted. Thus, the Examiner's argument that it is well-known that encryption of a packet would prevent subsequent components from processing the header of the packet is incorrect.

However, Examiner respectfully disagrees with PO's assertion. As explained earlier, under the broadest reasonable interpretation of the claim language, Kerr discloses a "plurality of components" for processing messages. For example, claim 1 of Kerr describes using a "plurality of devices" to apply "policy treatments" to a "plurality of messages," where policy treatments are used to perform "access control, ...security," "queuing," "accounting," "traffic profiling," etc. Id. at 10:27-40. Processing components can include "treatment with regard to switching," "access control," and "encryption." Id. at 4:20-34. "[S]pecial processing" can include "authentication" techniques "useful for implementing security 'firewalls.'" Id. at 35-46. Kerr further discloses that a "rewrite function" may be invoked "to alter the header for the packet." Id. at 4:55-62. These components can be used for "converting data with an input format into data with an output format," for example, the "encryption" and "rewrite" components to "alter" data to be processed. Id. at 4:30- 31, 4:55-62. The processing components of Kerr comprise "software routine" embodiments, as Kerr states that the processing instrumentality "may include specific hardware constructed or programmed performing the process steps described herein" or "a general purpose processor operating under program control." Id. at 2:51-55; see also Id. at Figs. 3-4 (illustrating software data structures).

Kerr discloses a "pointer" to the header rewrite "function" for a flow, which clearly implies that at least the header rewrite operation is embodied as a discrete software routine. Ex. 15 at 4:56-59. Moreover, one of ordinary skill in the art would understand the operations in Kerr would be organized as software routines in any event. While it can be practical to organize a short, simple program as a single software routine, those of ordinary skill in the art appreciate it is essentially impossible to build a large, complex piece of software such as Kerr without subdividing it into different software routines.

Discrete operations such as encryption and authentication are the quintessence of operations which would inevitably be organized as software routines by one of even minimal skill in the art. Thus, such organization is at least inherent in Kerr. Nevertheless, if it is determined that organizing the various operations as software routines is not disclosed or inherent, it was certainly at least obvious for one of ordinary skill to organize them as such.

When Kerr receives the first packet of a new flow, it determines the "proper treatment" for the packets of the flow. Kerr 4:12-15. It records an indication of this proper treatment in a flow record, so the proper treatment does not have to be determined again for subsequent packets of the flow (which would be needlessly inefficient). Kerr at 4:64-5:4. Depending on the flow, its "proper treatment" may include performing various distinct operations such as: "encryption"; header "rewrite"; "switching"; "access control"; "accounting"; and "authenticat[ion]." Kerr at 4:20-60. Not all operations are selected for all flows: i.e., sometimes determining the proper treatment "with regard to" an operation means not performing any form of the operation at all. See Kerr at 4:20-22. For example: (1) authentication is explicitly described as "special [non-standard] treatment" (Kerr at 4:35-39); (2) the pointer to a flow's header "rewrite" function can be either defined or "null" (Kerr at 4:56-59); and (3) the proper treatment "with regard to" encryption would be understood to include not performing any encryption or decryption at all on its packets (e.g., where the flow is directed to a destination outside a virtual private network, hence requiring no encryption or decryption by the router) (see Kerr at 4:21-32).

Determining the proper treatment may also include selecting a particular form of an operation. For example, flows in Kerr encompass one direction of a connection only. Kerr at 2:56 ("unidirectional stream"). Those of ordinary skill in the art understand that where a router is performing encryption on the packets of an outgoing flow, the router would also be performing decryption on the packets of the incoming flow for that connection.

As another example, Kerr explicitly teaches there is a "pointer" to the header "rewrite" function for a flow, rather than merely an on/off switch, clearly suggesting there could be several possible "rewrite" functions that might be selected. Kerr at 4:56-59. Whether disclosed or obvious, once it is determined these operations are embodied in software routines, it is clear Kerr would identify some of these routines but not others as the "proper treatment" for particular flows. For example, a first flow might be assigned an authentication routine and a decryption routine, while a second flow might be assigned a header rewrite routine and an encryption routine. And notably, this identification is clearly performed dynamically, after receiving the first packet. Kerr at 4:13-14 ("At a step 224, the routing device builds a new entry in the flow cache" and "determines proper treatment of packets.., in the message flow").

Moreover, one of ordinary skill in the art would appreciate that certain operations in Kerr must be performed in a certain order. E.g., rewriting or authenticating a packet's original header is impossible if it has already been encrypted. One of ordinary skill in the art would understand the necessity for such ordering, and arrange Kerr's operations accordingly. Thus, one of ordinary skill would understand the components must be arranged in a suitable sequential order, for Kerr to simply function as described. And in any event, it was certainly at least obvious to arrange them in a suitable sequential order, so that Kerr could function properly as described. Thus, Kerr discloses and renders obvious these claim elements.

In regards to PO's assertion regarding encrypting only IP packet contents and not the header, RFC 1825 explains there are "two modes" for performing encryption:

(1) "Tunnel-mode," which encrypts and **"encapsulates an entire IP datagram"** (including its IP header). Ex. 26 at 9. **PO argues this mode is impossible.** Response at 21 ("the packet header.., is never encrypted, as it must remain a well-formatted IP header in order for other routers to accept it and continue the routing process").

(2) "Transport-mode," which encrypts and "encapsulates" only "an upper-layer [transport] protocol (for example UDP or TCP)." Ex. 26 at 9. This is the mode described by PO in its Response. Response at 21 ("encryption takes place in the transport layer and above").

A router operating in Tunnel-mode encrypts the entire original IP packet from a host (including the packet's original destination), and prepends a new, second header to the packet. This second header can be used to address the packet to a second router (as if through a "tunnel") across an untrusted backbone. See Ex. 26 at 4-5, 9. When the second router receives the packet, it decrypts the original packet in its entirety, examines its first, original header, and forwards the packet to its original destination. Two distinct headers (one encrypted, one not) permit delivery to two distinct destinations along the route (first to another router, then to an end host). See Ex. 26 at

4-5, 9. This is the classic VPN design, and those of ordinary skill would certainly understand that Tunnel-mode encryption (as opposed to Transport-mode) would be used by for encryption to provide "virtual private network" feature. In contrast, Transport-mode is typically used for only the special case of "host-to-host encryption," where there is no intervening gateway "present in the connection" that would perform encryption for hosts as a service. See Ex. 26 at 4-5, 9. Therefore, PO's assertion that, encryption happens only in IP packet contents, and not the header, is not entirely true.

Therefore, Examiner respectfully disagrees with PO's assertion that Kerr teaches away from the input/output compatibility requirement, and the Office Action includes technical inaccuracies, while providing no "reason" or "evidence" showing that one of skill in the art would add such compatibility to Kerr; there is no *prima facie* case of obviousness.

PO argues in pages 31-32:

> Claim 1 requires "for each of a plurality of components in the identified non-predefined sequence, [1] retrieving state information relating to performing the processing of the component with the previous packet of the message; [2] performing the processing of the identified component with the packet and the retrieved state information; and [3] storing state information relating to the processing of the component with packet for use when processing the next packet

> of the message." In the Related Litigation, the Court construed state information as "**information specific to a software routine for a specific message** that is not information related to an overall path." *Markman* Order at 14 (Ex. 2) (emphasis added). Kerr discloses none of the claimed requirements of "retrieving," "processing" and "storing" message-specific state information for the components.
>
> Just as he did with Decasper, the Examiner conflates the claim requirement for "storing an indication" of the path to avoid re-identifying it for subsequent packets (which appears above the state information limitations) and the state information requirements. This improper conflation appears on page 9 of the Office Action, where the separate claim requirements are lumped together. They are separate limitations and they must be treated as such.

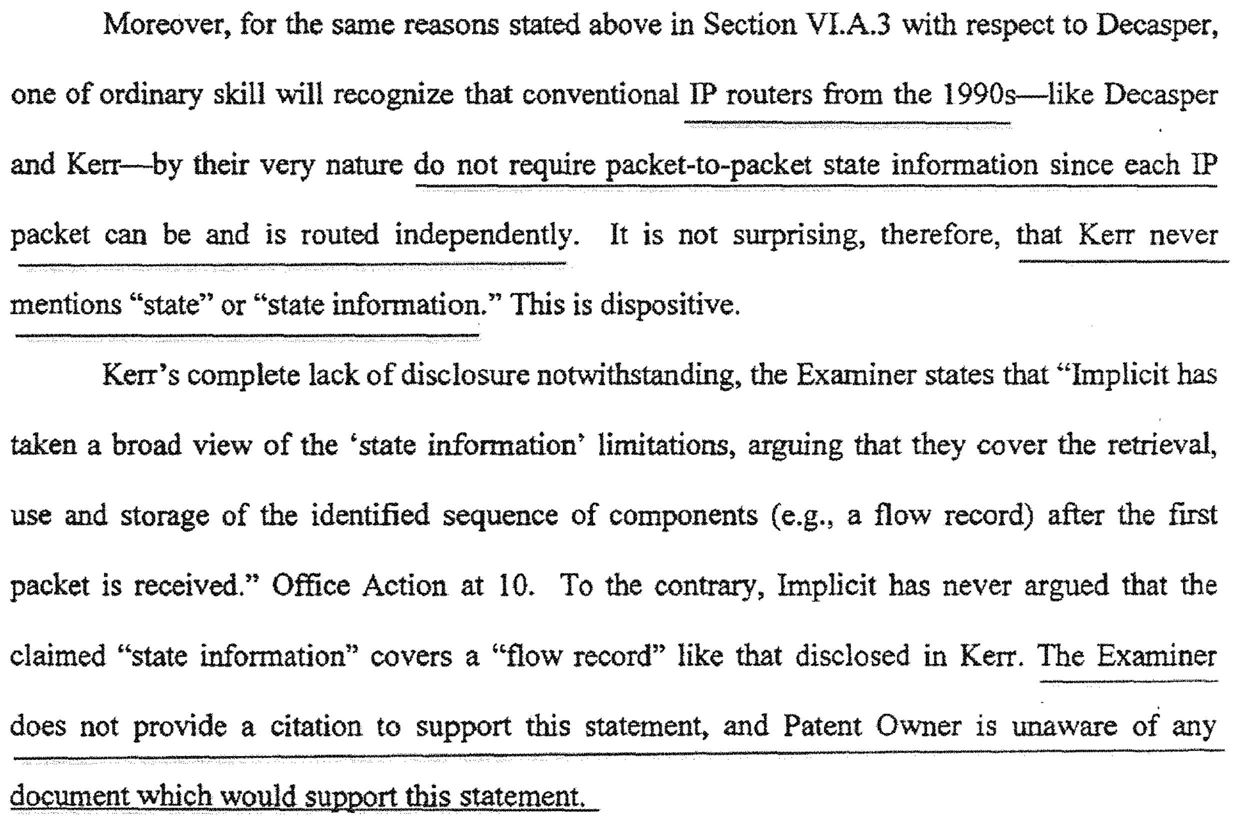
However, Examiner respectfully disagrees with PO's assertion. Examiner respectfully submits that "[I]n PTO reexamination, the standard of proof- a preponderance of evidence – is substantially lower than in a civil case" and there is no presumption of validity in reexamination proceedings." *678 F.3d 1357, 1364* (Fed. Cir. 2012) (internal citations omitted); see also *Old Reliable Wholesale, Inc. v. Cornell Corp., 635 F.3d 539, 548 n.6* (Fed. Cir. 2011) ("Whereas clear and convincing evidence is required to invalidate a patent in district court, a patent can be invalidated during PTO reexamination by a simple preponderance of the evidence."). "Claims are given 'their broadest reasonable interpretation, consistent with the specification, in reexamination proceedings.'" *In re Trans Texas Holding Corp., 498 F.3d 1290, 1298* (Fed. Cir. 2007) (quoting *In re Yamamoto, 740 F.2d 1569, 1571* (Fed. Cir. 1984). "Giving claims the broadest reasonable construction 'serves the public interest by reducing the possibility that claims, finally allowed, will be given broader scope than is justified.'" *In re Am. Acad. ofSci. Tech Ctr., 367 F.3d 1359, 1365* (Fed. Cir. 2004) (quoting Yamamoto, 740 F.2d at 1571). "Construing claims broadly during prosecution is not unfair to the applicant (or, in this case, the patentee), because the applicant has the opportunity to amend the claims to obtain more precise claim coverage." *In re Trans Texas Holding Corp., 498 F.3d 1290, 1298* (Fed. Cir. 2007) (citing *Yamamoto, 740 F.2d at 1571-72 and In re Zletz, 893 F.2d 319, 322* (Fed. Cir. 1989)). "The Board is required to use a different standard for construing claims than that used by district courts." *In re Am. Acad. ofSci. Tech Ctr., 367 F.3d 1359, 1369* (Fed. Cir. 2004). Indeed, the Federal . Circuit has repeatedly held that "it is error for the Board to 'apply the mode of claim interpretation that is used by courts in litigation, when interpreting the claims of issued patents in connection with determinations of infringement and validity.'" Id. (citing *Zletz, 893 F.2d at 321); see also In re Morris, 127 F.3d 1048, 1054* (Fed. Cir. 1997) ("It would be inconsistent with the role assigned to the PTO in issuing a patent to require it to interpret the claims in the same manner as judges who, post-issuance, operate under the assumption that the patent is valid)." Instead, the PTO is empowered and "obligated to give claims their broadest reasonable interpretation during examination." Id. Thus, in the instant case "the pending claims must be interpreted as broadly as their terms reasonably allow." *In re Zletz, 893 F.2d 321.*

In regards to alleged "conflating" accusition. Examiner submits that Examiner has no such intention during the articulation of the rejections. The rejection was articulated based on the claim construct and claim limitations as well as rejection proposed by the Third Party Requester's Request. Kerr clearly teaches:

storing an indication of each of the identified components so that the non-predefined sequence does not need to be re-identified for subsequent packets of the message; and for each of a plurality of packets of the message in sequence, for each of a plurality of components in the identified non-predefined sequence, retrieving state information relating to performing the processing of the component with the previous packet of the message; performing the processing of the identified component with the packet and the retrieved state information; and storing state information relating to the processing of the component with the packet for use when processing the next packet of the message (After receiving the first packet of a new flow, Kerr builds a new flow entry that is cached in memory, which constitutes **"storing"**. 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 (i.e.

satisfying the retrieving step) 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. 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 **(i.e. storing step)**, 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. 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. 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 **(i.e. processing step)**. 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).

PO argues in page 32:



Moreover, for the same reasons stated above in Section VI.A.3 with respect to Decasper, one of ordinary skill will recognize that conventional IP routers from the 1990s—like Decasper and Kerr—by their very nature do not require packet-to-packet state information since each IP packet can be and is routed independently. It is not surprising, therefore, that Kerr never mentions "state" or "state information." This is dispositive.

Kerr's complete lack of disclosure notwithstanding, the Examiner states that "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." Office Action at 10. To the contrary, Implicit has never argued that the claimed "state information" covers a "flow record" like that disclosed in Kerr. The Examiner does not provide a citation to support this statement, and Patent Owner is unaware of any document which would support this statement.

However, Examiner respectfully disagrees with PO's assertion. In regards to "state information", Examiner submits that the "accounting component": i.e., maintaining "a cumulative count for the number of packets" and "a cumulative count for the number of bytes" OA at 36, it is difficult to imagine how an accounting component could maintain these cumulative counts without, for each packet, retrieving the previous count, advancing it accordingly, and storing the result for use when processing the next packet of the message. As another example of a component that would maintain state information across packets, is the "encryption" component. OA at 10. Kerr supports "IP," and one of ordinary skill in the art be aware that according to the "Security Architecture for IP" (RFC 1825), an encryption algorithm which "MUST" be supported is the "ESP DEC-CBC" algorithm described in RFC 1829. Ex. 26 (RFC 1825) at 10, 21. RFC 1829 explains that to apply this mandatory encryption algorithm, "an Initialization Vector" must be placed in "[e]ach datagram" to be encrypted (i.e., in each packet)--and that a "common" technique for assigning these vectors "is simply a counter, beginning with a random chosen value." Ex. 27 (RFC 1829) at 1. Thus, an IP security plugin performing such encryption would, for each packet: retrieve the previous counter value; apply it to encrypt the packet; increment the counter value; and store it for use when encrypting the next packet. Given this standard background knowledge, this implementation would be assumed, and it was at least obvious.

PO argues in page 33:

Sticking with the improper "flow record" theme, the Office Action states that "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." Office Action at 10. But the "flow records" in Kerr relate to the "proper treatments," as explained above in Section VII.A.1. Those proper treatments concern the correct processing and are stored so that that the packets of the same flow can be processed with the same treatments, but this has nothing to do with state information about the components and the components' use of that state information, as required by the claims. Recognizing this deficiency and specifically referring to "state information on a component-by-component basis," the Office Action then refers to "components for access control, encryption, 'special treatment,' accounting, rewrite, among others." *Id.* Yet these are merely processes that can be performed and that might be part of the "proper treatments." Again, they have nothing to do with the claimed state information. Nor does the "timing information" referenced in the Office Action have anything to do with the claimed state information, in part, because it is unrelated to processing packets of a message. Instead, the timing information is strictly about expiring flow entries after they have no activity.

Additionally, for the reasons discussed in Section VII.A.1 above, Kerr does not disclose the identification of a "sequence of components." Instead, it teaches that routing devices "determine[] proper treatment of packets" and "enter[] information regarding such proper treatment in a data structure." Accordingly, Kerr fails to disclose retrieving, using, or storing of "information specific to a software routine for a specific message" "relating to performing the processing of the component," as required by claim 1.

However, Examiner respectfully disagrees with PO's assertion. As explained earlier, under the broadest reasonable interpretation of the claim language, Kerr discloses a "plurality of components" for processing messages. For example, claim 1 of Kerr describes using a "plurality of devices" to apply "policy treatments" to a "plurality of messages," where policy treatments are used to perform "access control, ...security," "queuing," "accounting," "traffic profiling," etc. Id. at 10:27-40. Processing components can include "treatment with regard to switching," "access control," and "encryption." Id. at 4:20-34. "[S]pecial processing" can include "authentication" techniques "useful for implementing security 'firewalls.'" Id. at 35-46. Kerr further discloses that a "rewrite function" may be invoked "to alter the header for the packet." Id. at 4:55-62. These components can be used for "converting data with an input format into data with an output format," for example, the "encryption" and "rewrite" components to "alter" data to be processed. Id. at 4:30- 31, 4:55-62. The processing components of Kerr comprise "software routine" embodiments, as Kerr states that the processing instrumentality "may include specific hardware constructed or programmed performing the process steps described herein" or "a general purpose processor operating under program control." Id. at 2:51-55; see also Id. at Figs. 3-4 (illustrating software data structures).

Examiner submits that the "accounting component": i.e., maintaining "a cumulative count for the number of packets" and "a cumulative count for the number of bytes" OA at 36, it is difficult to imagine how an accounting component could maintain these cumulative counts without, for each packet, retrieving the previous count, advancing it accordingly, and storing the result for use when processing the next packet of the message. As another example of a component that would maintain state information across packets, is the "encryption" component. OA at 10. Kerr supports

"IP," and one of ordinary skill in the art be aware that according to the "Security Architecture for IP" (RFC 1825), an encryption algorithm which "MUST" be supported is the "ESP DEC-CBC" algorithm described in RFC 1829. Ex. 26 (RFC 1825) at 10, 21. RFC 1829 explains that to apply this mandatory encryption algorithm, "an Initialization Vector" must be placed in "[e]ach datagram" to be encrypted (i.e., in each packet)--and that a "common" technique for assigning these vectors "is simply a counter, beginning with a random chosen value." Ex. 27 (RFC 1829) at 1. Thus, an IP security plugin performing such encryption would, for each packet: retrieve the previous counter value; apply it to encrypt the packet; increment the counter value; and store it for use when encrypting the next packet. Given this standard background knowledge, this implementation would be assumed, and it was at least obvious.

Kerr discloses a "pointer" to the header rewrite "function" for a flow, which clearly implies that at least the header rewrite operation is embodied as a discrete software routine. Ex. 15 at 4:56-59. Moreover, one of ordinary skill in the art would understand the operations in Kerr would be organized as software routines in any event. While it can be practical to organize a short, simple program as a single software routine, those of ordinary skill in the art appreciate it is essentially impossible to build a large, complex piece of software such as Kerr without subdividing it into different software routines. Discrete operations such as encryption and authentication are the quintessence of operations which would inevitably be organized as software routines by one of even minimal skill in the art. Thus, such organization is at least inherent in Kerr. Nevertheless, if it is determined that organizing the various operations as software routines is not disclosed or inherent, it was certainly at least obvious for one of ordinary skill to organize them as such.

When Kerr receives the first packet of a new flow, it determines the "proper treatment" for the packets of the flow. Kerr 4:12-15. It records an indication of this proper treatment in a flow record, so the proper treatment does not have to be determined again for subsequent packets of the flow (which would be needlessly inefficient). Kerr at 4:64-5:4. Depending on the flow, its "proper treatment" may include performing various distinct operations such as: "encryption"; header "rewrite"; "switching"; "access control"; "accounting"; and "authenticat[ion]." Kerr at 4:20-60. Not all operations are selected for all flows: i.e., sometimes determining the proper treatment "with regard to" an operation means not performing any form of the operation at all. See Kerr at 4:20-22. For example: (1) authentication is explicitly described as "special [non-standard] treatment" (Kerr at 4:35-39); (2) the pointer to a flow's header "rewrite" function can be either defined or "null" (Kerr at 4:56-59); and (3) the proper treatment "with regard to" encryption would be understood to include not performing any encryption or decryption at all on its packets (e.g., where the flow is directed to a destination outside a virtual private network, hence requiring no encryption or decryption by the router) (see Kerr at 4:21-32).

Determining the proper treatment may also include selecting a particular form of an operation. For example, flows in Kerr encompass one direction of a connection only. Kerr at 2:56 ("unidirectional stream"). Those of ordinary skill in the art understand that where a router is performing encryption on the packets of an outgoing flow, the router would also be performing decryption on the packets of the incoming flow for that connection.

As another example, Kerr explicitly teaches there is a "pointer" to the header "rewrite" function for a flow, rather than merely an on/off switch, clearly suggesting there could be several possible "rewrite" functions that might be selected. Kerr at 4:56-59. Whether disclosed or obvious, once it is determined these operations are embodied in software routines, it is clear Kerr would identify some of these routines but not others as the "proper treatment" for particular flows. For example, a first flow might be assigned an authentication routine and a decryption routine, while a second flow might be assigned a header rewrite routine and an encryption routine. And notably, this identification is clearly performed dynamically, after receiving the first packet. Kerr at 4:13-14 ("At a step 224, the routing device builds a new entry in the flow cache" and "determines proper treatment of packets.., in the message flow").

Moreover, one of ordinary skill in the art would appreciate that certain operations in Kerr must be performed in a certain order. E.g., rewriting or authenticating a packet's original header is impossible if it has already been encrypted. One of ordinary skill in the art would understand the necessity for such ordering, and arrange Kerr's operations accordingly. Thus, one of ordinary skill would understand the components must be arranged in a suitable sequential order, for Kerr to simply function as described. And in any event, it was certainly at least obvious to arrange them in a suitable sequential order, so that Kerr could function properly as described. Thus, Kerr discloses and renders obvious these claim elements.

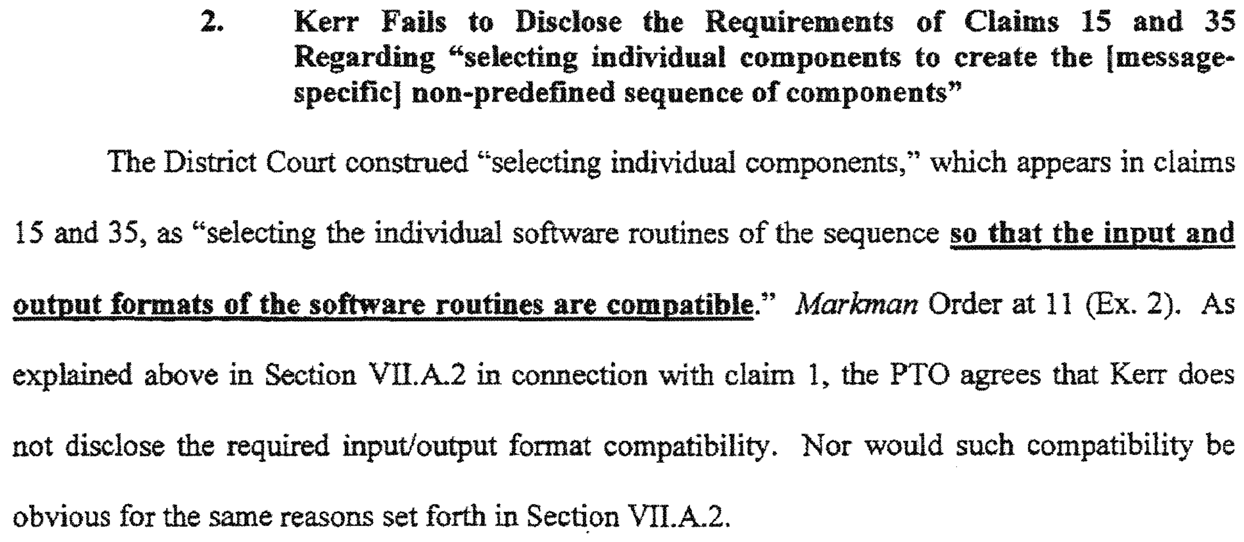
Therefore, Examiner respectfully disagrees with PO's assertion that Kerr does not disclose the claimed state information; Kerr does not even mention state or state information; Kerr does not teach any of the state information requirements "in as complete detail as is contained in" claim 1, as required by the MPEP.

PO argues in page 34:

> **1. Kerr Fails to Disclose the Requirements in Claims 15 and 35 Regarding Dynamically Identifying a Non-Predefined Sequence of Components**
>
> Like claim 1, claims 15 and 35 require "dynamically identifying" a "non-predefined sequence of components." The District Court construed "non-predefined sequence of components" as "a sequence of software routines that was not identified before the first packet of a message was received." *Markman* Order at 6 (Ex. 2). For the same reasons set forth above in Section VII.A.1 with respect to claim 1, Kerr also fails to disclose these requirements in claims 15 and 35.

Examiner respectfully disagrees with PO's assertion. Examiner has shown above that all the cited limitations of claim 1 are taught by Kerr. Therefore, claims 15 and 35 are not patentable for the same reasons.

PO argues in page 34:



2. Kerr Fails to Disclose the Requirements of Claims 15 and 35 Regarding "selecting individual components to create the [message-specific] non-predefined sequence of components"

The District Court construed "selecting individual components," which appears in claims 15 and 35, as "selecting the individual software routines of the sequence **so that the input and output formats of the software routines are compatible**." *Markman* Order at 11 (Ex. 2). As explained above in Section VII.A.2 in connection with claim 1, the PTO agrees that Kerr does not disclose the required input/output format compatibility. Nor would such compatibility be obvious for the same reasons set forth in Section VII.A.2.

Examiner respectfully disagrees with PO's assertion. Examiner has shown above that all the cited limitations of claim 1 are taught by Kerr. Therefore, claims 15 and 35 are not patentable for the same reasons.

PO argues in pages 34-35:

3. Kerr Fails to Disclose the Requirements of Claims 15 and 35 Regarding "State Information"

Claim 15 requires that "for each packet of each message, performing the processing of the identified non-predefined sequence of components of the message wherein state information generated by performing the processing of a component for a packet is available to the component when the component processes the next packet of the message." Claim 35 requires that "for each packet of each message, invoking the identified non-predefined sequence of components in sequence to perform the processing of each component for the packet wherein each component saves message-specific state information so that that component can use the saved message-specific state information when that component performs its processing on the next packet of the message." For the same reasons set forth above in Section VII.A.3 with respect to claim 1, Kerr also fails to disclose these state information requirements in claims 15 and 35.

Examiner respectfully disagrees with PO's assertion. Examiner has shown above that all the cited limitations of claim 1 are taught by Kerr. Therefore, claims 15 and 35 are not patentable for the same reasons.

<u>PO argues in page 35:</u>

VIII. SECONDARY INDICIA EVIDENCE OF NON-OBVIOUSNESS

In addition to the above arguments showing that claims 1, 15 and 35 are non-obvious, Implicit presents the declaration of the inventor, attached as Exhibit 1. The declaration and the exhibits attached to the declaration further show that the claims are non-obvious.

PO argues in page 2 of Declaration under 37 CFR 1.132:

COMMERCIAL SUCCESS

7. Implicit's '163 patented invention manifested itself into various products sold to sophisticated customers. For example, Implicit built the code for the first tablet computer, the Intel Web Tablet, released in 2000; the code for a home gateway solution designed by Intel to be released in 2001; and likewise provided the platform support for the Intel Media Player, released in 2004. Implicit had contracts with many other big companies, too, including AMD, Raytheon, and Thompson.

8. The commercial successes of the '163 invention were due, in part, to the claimed features in the '163 patent, in particular claims 1, 15 and 35. Implicit was able to succeed in these contracts and commercial endeavors because of the following features: (1) dynamically identifying a non-predefined sequence of components after receiving the first packet of a message; (2) the dynamic identification is such that the output format of the components of the non-predefined sequence match the input format of the next component in the sequence; and (3) using message-specific state information for the components when processing the packets.

However, Examiner respectfully disagrees with PO's assertion.

An affidavit or declaration attributing commercial success to a product or process "constructed according to the disclosure and claims of [the] patent application" or other equivalent language does not establish a nexus between the claimed invention and the commercial success because there is no evidence that the product or process which has been sold corresponds to the claimed invention, or that whatever commercial success may have occurred is attributable to the product or process defined by the claims. *Ex parte Standish, 10 USPQ2d 1454, 1458 (Bd. Pat. App. & Inter. 1988).*

EWP Corp. v. Reliance Universal, Inc., 755 F.2d 898, 225 USPQ 20 (Fed. Cir. 1985) (evidence of licensing is a secondary consideration which must be carefully appraised as to its evidentiary value because licensing programs may succeed for reasons unrelated to the unobviousness of the product or process, e.g., license is mutually beneficial or less expensive than defending infringement suits)

To be pertinent to the issue of nonobviousness, the commercial success of devices falling within the claims of the patent must flow from the functions and advantages disclosed or inherent in the description in the specification. Furthermore, the success of an embodiment within the claims may not be attributable to improvements or modifications made by others. *In re Vamco Machine & Tool, Inc., 752 F.2d 1564, 224 USPQ 617* (Fed. Cir. 1985).

A patentee seeking to rely on commercial success must satisfy its burden of showing "both that there is commercial success, and that the thing (product or method) that is commercially successful is the invention disclosed and claimed in the patent." *Demaco Corp. v. F. Von Langsdorff Licensing Ltd., 851 F. 2d 1387* (Fed. Cir. 1988) (noting theterm "nexus" is often used in this context). The Balassanian Declaration does neither.

The Balassanian Declaration alleges that the alleged invention of the '163 patent "'manifested itself into various products sold to sophisticated customers," and then proceeds to list a number of alleged embodiments and "big companies" with which Implicit allegedly "had contracts." Balassanian Declaration ¶ 7. But, of course, merely entering into "contracts" with companies that are "big" and "sophisticated" does not by itself show commercial success, and the Declaration does not even attempt to support its claim of success in the marketplace with conventional indicia such as volumes sold or revenue pertaining to the alleged embodiments. Moreover, the Declaration fails to provide any evidence establishing that these products do in fact embody the patents-in-suit, much less that it is the claimed features of the ' 163 patent that made these products commercially successful. *Brown& Williamson Tobacco, Corp. v. Phillip Morris, Inc., 229 F.3d 1120, 1130* (Fed.Cir. 2000) (nexus "must explain how the product's commercial success was caused, at least in part, by the claimed invention and not by the economic and commercial factors unrelated to the technical quality of the patented subject matter.").

The Balassanian Declaration thus fails to establish commercial success of the claimed invention.

PO argues in pages 3-4 of Declaration under 37 CFR 1.132:

LONG FELT BUT UNRESOLVED NEED

9. As the Internet changed in the late 1990's, the prior non-modular, pre-baked network operating systems were no longer workable. There is significant commentary in the trade press about the need for a different and better solution.

10. For example, the Scout reference's approach was discussed extensively in the prior *ex parte* reexamination. (A copy of the Scout dissertation was attached to Juniper Network's reexamination request.) Commentators referred to the Scout approach as "static, i.e., most configuration happens at build time, and runtime reconfiguration is impossible." Ex. A to this declaration at IMP127910. Dr. Larry Peterson was a Professor at the University of Arizona and was the advisor for David Mosberger, who wrote the Scout dissertation. Dr. Peterson later discussed and contrasted the '163 invention, which was called "Strings," in a paper dated June 1, 2001. Ex. B to this declaration. Dr. Peterson stated that "Strings defines a whole new paradigm." Ex. B at IMP127854. He then references the "dynamic configuration" aspect of Strings, which is a direct reference to the dynamic identification claimed in the '163 patent. *Id.*

However, Examiner respectfully disagrees with PO's assertion. Establishing long-felt need requires objective evidence that an art recognized problem existed in the art for a long period of time without solution. The relevance of long-felt need and the failure of others to the issue of obviousness depends on several factors. First, the need must have been a persistent one that was recognized by those of ordinary skill in the art. *In re Gershon, 372 F.2d 535, 539, 152 USPQ 602, 605* (CCPA 1967) ("Since the alleged problem in this case was first recognized by appellants, and others apparently have not yet become aware of its existence, it goes without saying that there could not possibly be any evidence of either a long felt need in the . . . art for a solution to a problem of dubious existence or failure of others skilled in the art who unsuccessfully attempted to solve a problem of which they were not aware."); *Orthopedic Equipment Co., Inc. v. All Orthopedic Appliances, Inc., 707 F.2d 1376, 217 USPQ 1281* (Fed. Cir. 1983) (Although the claimed invention achieved the desirable result of reducing inventories, there was no evidence of any prior unsuccessful attempts to do so.).

Second, the long-felt need must not have been satisfied by another before the invention by applicant. *Newell Companies v. Kenney Mfg. Co., 864 F.2d 757, 768, 9 USPQ2d 1417, 1426* (Fed. Cir. 1988) (Although at one time there was a long-felt need for a "do-it-yourself" window shade material which was adjustable without the use of tools, a prior art product fulfilled the need by using a scored plastic material which could be torn. "[O]nce another supplied the key element, there was no long-felt need or, indeed, a problem to be solved".)

Third, the invention must in fact satisfy the long-felt need. *In re Cavanagh, 436 F.2d 491, 168 USPQ 466* (CCPA 1971).

Examiner agrees with the Third Party (Response pages 41-42): The Balassanian Declaration also fails to prove that there was a long left but unresolved need in the field that was somehow satisfied by the claimed invention. See *In re Cavanagh, 436 F.2d 491,495* (C.C.P.A. 1971). The Balassanian Declaration claims there was "significant commentary" on this topic but cites only two references: (1) a single sentence from an almost 200-page dissertation that makes no mention of PO's technology and (2) a short paper of unstated origin or authorship regarding an PO product known as "Strings." Balassanian Declaration ¶ 10. As to the first of these, although the cited sentence asserts that "runtime reconfiguration is impossible" with respect to the "Scout" prior art product, it does support this statement, and never states that Kerr or Decasper98 or any number of other existing technologies lacked this feature. Nor does the Balassanian Declaration explain how "runtime reconfiguration" has anything to do with the claims of the '163 patent, especially after the District Court in the concurrent litigation rejected PO's attempt to construe the claims to cover sequences of components that were merely "changeable at runtime." App. R9 (Claim Construction Order) at 6. Without proof of nexus to the patent claims, PO's reliance on this dissertation fails.

Similarly, the second cited reference makes a vague statement about PO's alleged "Strings" embodiment "defin[ing] a whole new paradigm," but fails to connect that statement to anything relating to the claimed invention. The Balassanian Declaration again fails to provide any evidence establishing that the Strings product actually embodied the '163 patent. *Geo M. Martin Co. v.Alliance Mach. sys. Int 'l LLC, 618 F.3d 1294, 1305* (Fed. Cir. 2010) ("Industry praise must be linked to the patented invention."). And although the reference suggests at most that one other particular prior art technology (object-oriented programming) may not have necessarily supported "dynamic configuration.., at runtime," it does not explain how this is relevant to the claims as properly construed (i.e., after the District Court rejected INI's proposed "runtime" construction). Nor does the reference state that Kerr or Decapser98 or other known technologies lack this characteristic.

Finally, and perhaps most fundamentally, INI provides virtually none of the context needed to properly evaluate the paper for purposes of a non-obviousness analysis--for example, whether this was an independent, scholarly or peer-reviewed article as opposed to a paid piece commissioned by PO for use in marketing.

In short, as shown above, there is no evidence that PO's claimed embodiment

ever

experienced any commercial success, much less solved any problems or satisfied any long felt needs in the field. *Cavanagh, 436 F.2d at 495*. Thus, this argument also fails.

<u>PO argues in page 4 of Declaration under 37 CFR 1.132:</u>

11. Implicit has filed patent infringement lawsuits involving the '163 patent, among others. These lawsuits include:

- *Implicit Networks, Inc. v. Advanced Micro Devices, Inc., Intel Corporation, NVIDIA Corporation, Raza Microelectronics, RealNetworks, Inc., and SunMicrosystems, Inc.*, C08-0184 JLR, U.S. District Court for the Western District of Washington (asserting infringement of the '163 patent).
- *Implicit Networks, Inc. v. Microsoft Corporation*, CV09-5628, U.S. District Court for the Northern District of California (asserting infringement of the '163 patent).
- *Implicit Networks, Inc. v. Cisco Systems, Inc.*, C10-3606, U.S. District Court for the Northern District of California (asserting infringement of the '163 patent and the '857 patent).

- *Implicit Networks, Inc. v. Citrix Systems, Inc.*, C10-3766, U.S. District Court for the Northern District of California (asserting infringement of the '163 patent).

12. As part of the resolution of the lawsuits involving the '163 patent, numerous companies have taken licenses under the '163 patent. These licenses also extend to other patents in the Implicit portfolio (see Implicit's *Markman* Brief referencing 13 patents). The licenses concerning the '163 patent (which now number over 10 licenses) include such companies as: Microsoft, IBM, Oracle, Sun, Cisco, and Citrix. See Implicit's *Markman* Brief at 6.

Examiner respectfully disagrees with PO's assertion.

EWP Corp. v. Reliance Universal, Inc., 755 F.2d 898, 225 USPQ 20 (Fed. Cir. 1985) (evidence of licensing is a secondary consideration which must be carefully appraised as to its evidentiary value because licensing programs may succeed for reasons unrelated to the unobviousness of the product or process, e.g., license is mutually beneficial or less expensive than defending infringement suits)

Examiner agrees with the Third Party (Response page 42):

Finally, the Balassanian Declaration fails to show any acquiescence through licensing that would tend to show non-obviousness. PO is relying here on a number of licenses reached as a result of litigation. Balassanian Declaration ¶ 11. PO has expressly admitted in the District Court proceedings that these licenses were "litigation related conlpromises." App. R11 (PO Responses to Requests for Admission) at 12. Without attaching the licenses themselves or undertaking any analysis of the terms and amounts paid (and why), the Examiner cannot be expected to fairly evaluate whether such licenses reflect acknowledgements of the value of the alleged invention, or mere payments to avoid the burdens and costs of litigation. Moreover, the Balassanian Declaration fails to present proof that any of the licensed products actually fall within the scope of the claims at issue (e.g., by admission of the licensors or otherwise). Thus, these licenses do not and cannot demonstrate non-obviousness.

Claim Rejections - 35 USC § 102

5. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> (b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

> A person shall be entitled to a patent unless –

> (e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

Claim Rejections - 35 USC § 103

6. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

7. The factual inquiries set forth in *Graham* v. *John Deere Co.*, 383 U.S. 1, 148 USPQ 459 (1966), that are applied for establishing a background for determining obviousness under 35 U.S.C. 103(a) are summarized as follows:

1. Determining the scope and contents of the prior art.
2. Ascertaining the differences between the prior art and the claims at issue.
3. Resolving the level of ordinary skill in the pertinent art.
4. Considering objective evidence present in the application indicating obviousness or nonobviousness.

8. Claim 1 is rejected under 35 U.S.C. 103(a) as being unpatentable over Kerr et al. (US PAT 6243667, hereinafter Kerr).

In regards to claim 1, Kerr discloses ***a method in a computer system*** (column 2, lines 30-32, However, those skilled in the art would recognize, after perusal of this application, that embodiments of the invention may be implemented using a set of general purpose computers operating under program control, and that modification of a set of general purpose computers to implement the process steps and data structures described herein would not require undue invention) ***for processing a message having a sequence of packets*** (column 1 lines 59-60, The invention provides a method and system for switching in networks responsive to message flow patterns. A message "flow" is defined to comprise a set of packets to be transmitted between a particular source and a particular destination. When routers in a network identify a new message flow, they determine the proper processing for packets in that message flow and cache that information for that message flow. Thereafter, when routers in a network identify a packet which is part of that message flow, they process that packet according to the proper processing for packets in that message flow. The proper processing may include a determination of a destination port for routing those packets and a determination of whether access control permits routing those packets to their indicated destination) ***the method comprising:***

providing a plurality of components, each component being a software routine for converting data with an input format into data with an output format (Kerr discloses a "plurality of components" for processing messages. For example, claim 1 of Kerr describes using a "plurality of devices" to apply "policy treatments" to a "plurality of messages," where policy treatments are used to perform "access control, ...security," "queuing," "accounting," "traffic profiling," etc. Id. at 10:27-40. Processing components can include "treatment with regard to switching," "access control," and "encryption." Id. at 4:20-34. "[S]pecial processing" can include "authentication" techniques "useful for implementing security 'firewalls.'" Id. at 35-46. Kerr further discloses that a "rewrite function" may be invoked "to alter the header for the packet." Id. at 4:55-62. These components can be used for "converting data with an input format into data with an output format," under Implicit' s apparent claim constructions, for example (as described above), the "encryption" and "rewrite" components to “alter" data to be processed. Id. at 4:30- 31, 4:55-62. The processing components of Kerr comprise "software routine" embodiments, as Kerr states that the processing instrumentality "may include specific hardware constructed or programmed performing the process steps described herein" or "a general purpose processor operating under program control." Id. at 2:51-55; see also id. at Figs. 3-4 (illustrating software data structures));

for the first packet of the message, dynamically identifying a non-predefined sequence of components for processing the packets of the message wherein dynamically identifying includes selecting individual components to create the non-predefined sequence of components after the first packet is received (claim 1, lines 31-40, column 4 lines 20-34, column 3 line 38-column 6 line 27 identifying a first one message of a first plurality of messages associated with an application layer, said first plurality of messages having at least one policy treatment in common, said first plurality of messages being identified in response to an address of a selected source device and an address of a selected destination device, wherein said policy treatment comprises at least one of the access control information, security information, queuing information, accounting information, traffic profiling information, and policy information; In a preferred embodiment, the proper treatment of packets 150 in the message flow 160 includes treatment with regard to switching (thus, the routing device 140 determines an output port for switching packets 150 in the message flow 160), with regard to access control (thus, the routing device 140 determines whether packets 150 in the message flow 160 meet the requirements of access control, as defined by access control lists in force at the routing device 140), with regard to accounting (thus, the routing device 140 creates an accounting record for the message flow 160), with regard to encryption (thus, the routing device 140 determines encryption treatment for packets 150 in the message flow 160), and any special treatment for packets 150 in the message flow 160. FIG. 2 shows a method for routing in networks responsive to message flow patterns. In broad overview, the method for routing in networks responsive to message flow patterns comprises two parts. In a first part, the routing device 140 builds and uses a flow cache described in further detail with regard to FIG. 3), in which routing information to be used for packets 150 in each particular message flow 160 is recorded and from which such routing information is retrieved for use...A method 200 for routing in networks responsive to message flow patterns is performed by the routing device 140. At a flow point 210, the routing device 140 is disposed for building and using the flow cache. 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. At a step 223, the routing device 140 performs a lookup in a flow cache for the identified message flow 160. If the lookup is unsuccessful, the identified message flow 160 is a "new" message flow 160, and the routing device 140 continues with the step 224. If the lookup is successful, the identified message flow 160 is an "old" message flow 160, and the routing device 140 continues with the step 225. In a preferred embodiment, the routing device 140 determines a hash table key responsive to the flow key 310. This aspect of the step 223 is described in further detail with regard to FIG. 3. At a step 224, the routing device

140 builds a new entry in the flow cache. The routing device 140 determines proper treatment of packets 150 in the message flow 160 and enters information regarding such proper treatment in a data structure pointed to by the new entry in the flow cache. In a preferred embodiment, the routing device 140 determines the proper treatment by performing a lookup in an IP address cache as shown in FIG. 4. In a preferred embodiment, the proper treatment of packets 150 in the message flow 160 includes treatment with regard to switching (thus, the routing device 140 determines an output port for switching packets 150 in the message flow 160), with regard to access control (thus, the routing device 140 determines whether packets 150 in the message flow 160 meet the requirements of access control, as defined by access control lists in force at the routing device 140), with regard to accounting (thus, the routing device 140 creates an accounting record for the message flow 160), with regard to encryption (thus, the routing device 140 determines encryption treatment for packets 150 in the message flow 160), and any special treatment for packets 150 in the message flow 160. In a preferred embodiment, the routing device 140 performs any special processing for new message flows 160 at this time...Thereafter, the routing device 140 proceeds with the step 225, using the information from the new entry in the flow cache, just as if the identified message flow 160 were an "old" message flow 160 and the lookup in a flow cache had been successful. At a step 225, the routing device 140 retrieves routing information from the entry in the flow cache for the identified message flow 160. In a preferred embodiment, the entry in the flow cache includes a pointer to a rewrite function for at least part of a header for the packet 150. If this pointer is non-null, the

routing device 140 invokes the rewrite function to alter the header for the packet 150. At a step 226, the routing device 140 routes the packet 150 responsive to the routing information retrieved at the step 225. 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. Thus, in a preferred embodiment, the routing device 140 routes the packet 150 to an output port, determines whether access is allowed for the packet 150, determines encryption treatment for the packet 150, and performs any special treatment for the packet 150, all responsive to information in the entry in the flow cache. In a preferred embodiment, the routing device 140 also enters accounting information in the entry in the flow cache for the packet 150. When routing each packet 150 in the message flow 160, the routing device 140 records the cumulative number of packets 150 and the cumulative number of bytes for the message flow 160. Because the routing device 140 processes each packet 150 in the message flow 160 responsive to the entry for the message flow 160 in the flow cache, the routing device 140 is able to implement administrative policies which are designated for each message flow 160 rather than for each packet 150);

and storing an indication of each of the identified components so that the non-predefined sequence does not need to be re-identified for subsequent packets of the message; and for each of a plurality of packets of the message in

sequence, for each of a plurality of components in the identified non-predefined sequence, retrieving state information relating to performing the processing of the component with the previous packet of the message; performing the processing of the identified component with the packet and the retrieved state information; and storing state information relating to the processing of the component with the packet for use when processing the next packet of the message (After receiving the first packet of a new flow, Kerr builds a new flow entry that is cached in memory, which constitutes "storing". 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. 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. 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. 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).

Kerr does not explicitly teach ***processing the packets of the message such that the output format of the components match the input format of the next component.***

Regarding the limitation "such that the output format of the components ... match the input format of the next component," it was well-known to those of ordinary skill in the art that certain operations on a packet must be performed in a certain order: e.g., if a packet is first converted into an encrypted format by a first component, a subsequent component would be unable to, e.g., rewrite its headers (because it was expecting to receive the packet in an unencrypted format). See id. at 4:31-32 ("encryption treatment for packets.., in the message flow"), 4:57-58 ("rewrite function for.., a header for the packet"). Thus, it was certainly at least obvious for one of ordinary skill in the art to arrange the sequence of components in a compatible manner, such that the output format of one matches the input format of the next-- rather than arranging them in an incompatible manner whereby various component(s) would be unable to perform their function(s).

9. Claim 1 is rejected under 35 U.S.C. 103(a) as being unpatentable over Decasper98.

In regards to claim 1, Decasper98 teaches ***a method in a computer system***

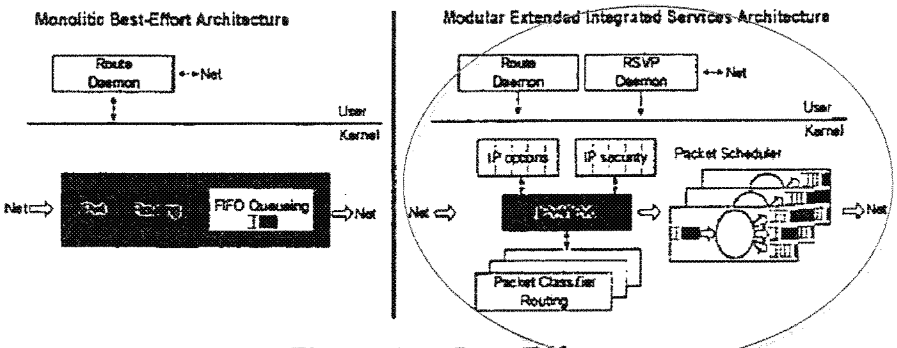
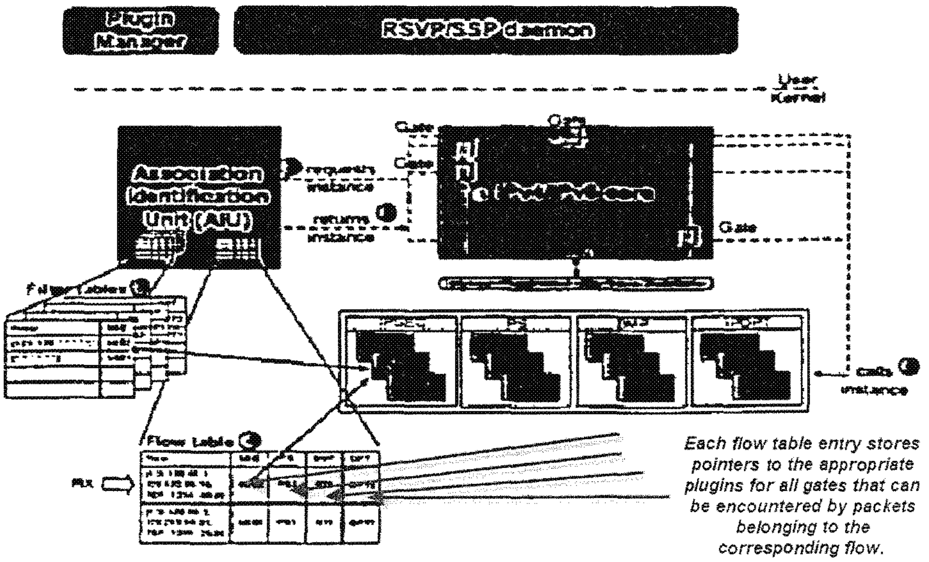


Figure 1. : Best Effort vs Extended Integrated Services Router (EISR)

for processing a message having a sequence of packets (Decasper98 explains: "it is very important to be able to quickly and efficiently classify packets into flows, and to apply different policies to different flows; these are both things that our architecture excels at doing." Ex. 25 at 2. Flows may represent "longer lived packet streams": Because the deployment of multimedia data sources and applications (e.g. real-time audio/video) will produce longer lived packet streams with more packets per session than is common in today's environment, an integrated services router architecture should support the notion of flows and build upon it. Id. at 3. A flow is defined as a group of packets which satisfy a specific filter. See id. at 3 ("Sets of flows are specified using filters .... Filters can also match individual end-to-end application flows"). Id. at 3. A flow would comprise a "message" under Implicit's apparent claim constructions. See section IV.C), ***the method comprising:***

providing a plurality of components (Decasper98 teaches that "[o]ne of the novel features of our design is the ability to bind different plugins to individual flows." Id. at 1), ***each component being a software routine for converting data*** (Id. at 2 ("plugins are kernel software modules that are.., responsible for performing certain functions on specified network flows.") for the first packet of the message, dynamically identifying a non-predefined sequence of components for processing the packets of the message wherein dynamically identifying includes selecting individual components to create the non-predefined sequence of components after the first packet is received (When the first packet of a new flow arrives, Decasper98 performs an expensive series of filter operations to determine the correct sequence of plugin components to be applied to the flow. See Ex. 25 at 5-6 ("The processing of the first packet of a new flow.., involves n filter table lookups to create a single entry in the flow table for the new flow."). This expensive series of filter operations does not need to be repeated for subsequent packets of the flow, because the new "entry... in the flow" table serves as a fast cache for future lookup of packets belonging to that flow," and the entry "stores pointers to the appropriate plugins." Id. at 5. Performance is thus enhanced for subsequent packets of the flow, since "[u]sually, filter table lookups are much slower than flow table lookups." Id. See also id. at 3 ("Subsequent packets get this information from a fast flow cache which temporarily stores the information gathered by processing the first packet."). Decasper98 assigns the sequence of plugins to the flow on the basis of lookups in multiple independent "filter tables." E.g., id. at 5-7 ("The processing of the first packet of a new flow.., involves n filter table lookup to create a single entry in the flow table for the new flow"); 7 ("multiple lookups (in different filter tables)"). E.g., a first filter table determines whether a first plugin is added to the sequence, a second independent filter

table determines whether a second plugin is added, a third independent filter table determines whether a third plugin is added, and so on. See Id. at 5-7.



Each flow table entry stores pointers to the appropriate plugins for all gates that can be encountered by packets belonging to the corresponding flow.

Figure 3. : System Architecture and Data Path

In the AIU, all flows start out being uncached (i.e., they do not have an entry in the flow table). If an incoming packet belongs to an uncached flow, its lookup in the flow table data structure will fail (i.e., there is a cache miss). In this case, the packet needs to be looked up in a different data structure that we call a **filter table**. Filter tables store the bindings between filters and plugins for each gate. The filter table lookup algorithm finds the most specific matching filter (described later) that has been installed in the table, and returns the corresponding plugin instance. 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. The processing of the first packet of a new flow with *n* gates involves *n* filter table lookups to create a single entry in the flow table for the new flow.

);

for the first packet of the message, dynamically identifying a non-predefined sequence of components for processing the packets of the message wherein dynamically identifying includes selecting individual components to create the non-predefined sequence of components after the first packet is received (when the first packet of a new flow arrives, Decasper98 performs an expensive series of filter operations to determine the correct sequence of plugin components to be applied to the flow. See Ex. 25 at 5-6 ("The processing of the first packet of a new flow.., involves n filter table lookups to create a single entry in the flow table for the new flow."). This expensive series of filter operations does not need to be repeated for subsequent packets of the flow, because the new "entry... in the flow" table serves as a fast cache for future lookup of packets belonging to that flow," and the entry "stores pointers to the appropriate plugins." Id. at 5. Performance is thus enhanced for subsequent packets of the flow, since "[u]sually, filter table lookups are much slower than flow table lookups." Id. See also id. at 3 ("Subsequent packets get this information from a fast flow cache which temporarily stores the information gathered by processing the first packet.").

Decasper98 assigns the sequence of plugins to the flow on the basis of lookups in multiple independent "filter tables." E.g., id. at 5-7 ("The processing of the first packet of a new flow.., involves n filter table lookup to create a single entry in the flow table for the new flow"); 7 ("multiple lookups (in different filter tables)"). E.g., a first filter table determines whether a first plugin is added to the sequence, a second independent filter

table determines whether a second plugin is added, a third independent filter table determines whether a third plugin is added, and so on. See id. at 5-7.

This leads "exponentially" to an enormous number of possible sequences that might be applied to the first packet of a flow when it arrives, "even with very few installed filters." See Id. at 7.2. These various possible sequences are not stored or enumerated anywhere in the system ahead of time. Instead, the sequence of plugins for a flow is generated algorithmically when the first packet of a flow arrives, by applying a series of filter operation to packet data which was not available to the system until that moment. See id. at 5-7.

Decasper98 explicitly considers and rejects a "theoretically possible" alternative approach, which is to replace this system of multiple independent filters with "a single global filter table." Id. at 7. Under this alternative approach, only a single filter would apply to a particular flow, and that single filter would specify the entire sequence of components to be applied to it. See Id. When the first packet arrived, the system would find the single matching filter and then essentially just read off the sequence of components to be applied to that flow. See Id. Thus, the sequence would be pre-defined and readily identifiable as such in a specific filter entry, even before the first packet arrived.

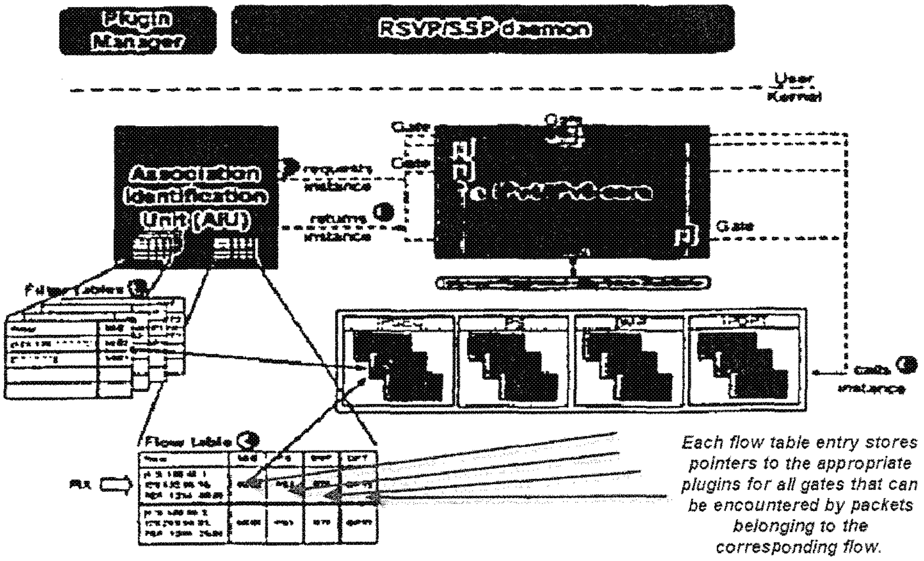
However, Decasper98 rejects this approach as "practically infeasible because the space requirements for the global table can, even with very few installed filters, increase very quickly (exponentially) to unacceptable levels." Id. In other words, Decasper98's multiple filter table approach implies so many potential valid sequences that it is impossible to even enumerate them all ahead of time in memory--since they would not fit.

Instead, Decasper98 adopts an algorithmic approach where the correct sequence is generated dynamically on demand, by applying the series of multiple filters to the first packet when it arrives. Thus, under Implicit's apparent claim constructions, Decasper98 discloses "for the first packet of the message, dynamically identifying a non-predefined sequence of components for processing the packets of the message."

Decasper98 discloses this "dynamically identifying" claim element under Implicit's apparent claim constructions. Decasper98 also teaches "selecting individual components to create the non-predefined sequence of components after the first packet is received." As explained above, after the first packet of a flow arrives, Descapser98 applies a series of independent filters to it, each of which may select a different individual plugin. Id at 5-7. See also, e.g., id. at 4 (Figure 2, showing various individual plugins that might be selected within each category, e.g., "BMP1 BMP2 BMP3"). The very purpose of this architecture is to apply the fight specific individual plugins in a tailored manner to each particular flow. E.g., id. at 2 ("it is very important to be able to quickly and efficiently classify packets into flows, and to apply different policies to different flows"), 3, 7)

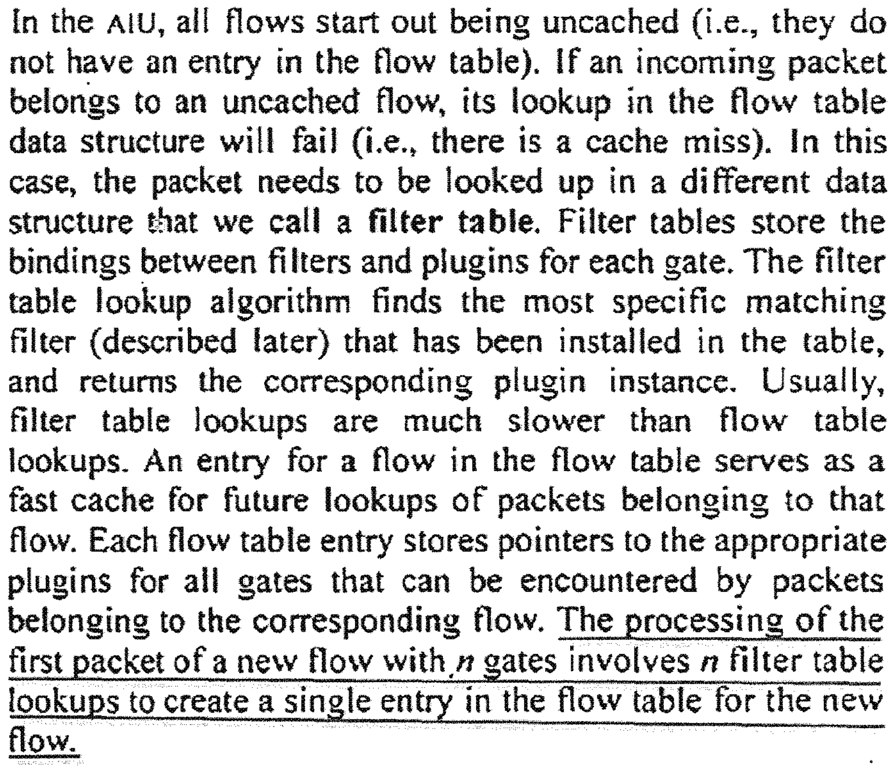
and storing an indication of each of the identified components so that the non-predefined sequence does not need to be re-identified for subsequent packets of the message; and for each of a plurality of packets of the message in sequence, for each of a plurality of components in the identified non-predefined

sequence, retrieving state information (pointers) *relating to performing the processing of the component with the previous packet of the message; performing the processing of the identified component with the packet and the retrieved state information; and storing state information relating to the processing of the component with the packet for use when processing the next packet of the message* (



Each flow table entry stores pointers to the appropriate plugins for all gates that can be encountered by packets belonging to the corresponding flow.

Figure 3. : System Architecture and Data Path



In the AIU, all flows start out being uncached (i.e., they do not have an entry in the flow table). If an incoming packet belongs to an uncached flow, its lookup in the flow table data structure will fail (i.e., there is a cache miss). In this case, the packet needs to be looked up in a different data structure that we call a **filter table**. Filter tables store the bindings between filters and plugins for each gate. The filter table lookup algorithm finds the most specific matching filter (described later) that has been installed in the table, and returns the corresponding plugin instance. 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. The processing of the first packet of a new flow with *n* gates involves *n* filter table lookups to create a single entry in the flow table for the new flow.

This cycle is executed only for the first packet arriving on an uncached flow. Subsequent packets follow a faster path because of the cached entry in the flow table. Note that in our system, we have created optimized implementations of both the flow and filter tables, allowing for high performance on both the cached and uncached paths. These implementations are described in Section 5.

Cached flow processing involves the following sequence:

- **Processing at the first gate:** When a packet from a cached flow encounters the **first** gate, the AIU is called to request the plugin instance. This time, the pointer to the instance requested is already in the flow table. The flow table is looked up efficiently, and the plugin instance pointer corresponding to the calling gate is returned. No filter table lookups are required.
- **Associating the packet with a flow index:** Together with the instance requested, the AIU returns a pointer to the row in the flow table where the information associated with the flow is stored. This pointer is called the flow index (FIX), and is stored in the packet's mbuf[1]. The instance is then called to process the packet, following which the IP stack passes the packet on to the next gate.
- **Processing at subsequent gates:** Once the packet has made its way past the first gate, the AIU does not have to be called upon to classify the packets at the remaining gates. Macros implementing a gate can retrieve the instance pointers cached in the flow table by accessing the FIX stored in the packet. This allows us to pass packets to the appropriate instances in a very efficient manner using an indirect function call instead of a "hardwired" function call. We show in section 7 that this does not imply significant performance penalties.

Decasper98 does not explicitly teach ***converting data with an input format into data with an output format; processing the packet of the message such that the output format matches the input format of the next component***.

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., process any IPv6 option headers in the packet, or to insert any new ones (because it was expecting to receive the packet in an unencrypted format). Thus, it was certainly obvious for one of ordinary skill in the art to arrange the sequence of components in a compatible manner, such that the output format of one matches the input format of the next.

10. Claims 15 and 35 are rejected under 35 U.S.C. 102(e) as being anticipated by Kerr et al. (US PAT 6243667, hereinafter Kerr).

In regards claim 15, Kerr anticipates ***a method in a computer system*** (column 2, lines 30-32, However, those skilled in the art would recognize, after perusal of this application, that embodiments of the invention may be implemented using a set of general purpose computers operating under program control, and that modification of a set of general purpose computers to implement the process steps and data structures described herein would not require undue invention) ***for demultiplexing packets of messages*** (column 1 lines 59-60, The invention provides a method and system for switching in networks responsive to message flow patterns. A message "flow" is defined to comprise a set of packets to be transmitted between a particular source and a particular destination. When routers in a network identify a new message flow, they determine the proper processing for packets in that message flow and cache that information for that message flow. Thereafter, when routers in a network identify a packet which is part of that message flow, they process that packet according to the proper processing for packets in that message flow. The proper processing may include a determination of a destination port for routing those packets and a determination of whether access control permits routing those packets to their indicated destination), ***the method comprising:***

dynamically identifying a non-predefined sequence of components for processing each message based on the first packet of the message so that subsequent packets of the message can be processed without re-identifying the components (claim 1, lines 31-40, column 4 lines 20-34, column 3 line 38-column 6 line 27 identifying a first one message of a first plurality of messages associated with an application layer, said first plurality of messages having at least one policy treatment in common, said first plurality of messages being identified in response to an address of a selected source device and an address of a selected destination device, wherein said policy treatment comprises at least one of the access control information, security information, queuing information, accounting information, traffic profiling information, and policy information; In a preferred embodiment, the proper treatment of packets 150 in the message flow 160 includes treatment with regard to switching (thus, the routing device 140 determines an output port for switching packets 150 in the message flow 160), with regard to access control (thus, the routing device 140 determines whether packets 150 in the message flow 160 meet the requirements of access control, as defined by access control lists in force at the routing device 140), with regard to accounting (thus, the routing device 140 creates an accounting record for the message flow 160), with regard to encryption (thus, the routing device 140 determines encryption treatment for packets 150 in the message flow 160), and any special treatment for packets 150 in the message flow 160. FIG. 2 shows a method for routing in networks responsive to message flow patterns. In broad overview, the method for routing in networks responsive to message flow patterns comprises two parts. In a first part, the routing device 140 builds and uses a flow cache described in further detail with regard to FIG. 3), in which routing information to be used for packets 150 in each particular message flow 160 is recorded and from which such routing information is retrieved for use...A method 200 for routing in networks responsive to message flow patterns is performed by the routing device 140. At a flow point 210, the routing device 140 is disposed for building and using the flow cache. 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. At a step 223, the routing device 140 performs a lookup in a flow cache for the identified message flow 160. If the lookup is unsuccessful, the identified message flow 160 is a "new" message flow 160, and the routing device 140 continues with the step 224. If the lookup is successful, the identified message flow 160 is an "old" message flow 160, and the routing device 140 continues with the step 225. In a preferred embodiment, the routing device 140 determines a hash table key responsive to the flow key 310. This aspect of the step 223 is described in further detail with regard to FIG. 3. At a step 224, the routing device 140 builds a new entry in the flow cache. The routing device 140 determines proper treatment of packets 150 in the message flow 160 and enters information regarding such proper treatment in a data structure pointed to by the new entry in the flow cache. In a preferred embodiment, the routing device 140 determines the proper treatment by performing a lookup in an IP address cache as shown in FIG. 4. In a preferred embodiment, the proper treatment of packets 150 in the message flow 160 includes treatment with regard to switching (thus, the routing device 140 determines an output port for switching packets 150 in the message flow 160), with regard to access control (thus, the routing device 140 determines whether packets 150 in the message flow 160 meet the requirements of access control, as defined by access control lists in force at the routing device 140), with regard to accounting (thus, the routing device 140 creates an accounting record for the message flow 160), with regard to encryption (thus, the routing device 140 determines encryption treatment for packets 150 in the message flow 160), and any special treatment for packets 150 in the message flow 160. In a preferred embodiment, the routing device 140 performs any special processing for new message flows 160 at this time...Thereafter, the routing device 140 proceeds with the step 225, using the information from the new entry in the flow cache, just as if the identified message flow 160 were an "old" message flow 160 and the lookup in a flow cache had been successful. At a step 225, the routing device 140 retrieves routing information from the entry in the flow cache for the identified message flow 160. In a preferred embodiment, the entry in the flow cache includes a pointer to a rewrite function for at least part of a header for the packet 150. If this pointer is non-null, the routing device 140 invokes the rewrite function to alter the header for the packet 150. At a step 226, the routing device 140 routes the packet 150 responsive to the routing information retrieved at the step 225. 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. Thus, in a preferred embodiment, the routing device 140 routes the packet 150 to an output port, determines whether access is allowed for the packet 150, determines encryption treatment for the packet 150, and performs any special treatment for the packet 150, all responsive to information in the entry in the flow cache. In a preferred embodiment, the routing device 140 also enters accounting information in the entry in the flow cache for the packet 150. When routing each packet 150 in the message flow 160, the routing device 140 records the cumulative number of packets 150 and the cumulative number of bytes for the message flow 160. Because the routing device 140 processes each packet 150 in the message flow 160 responsive to the entry for the message flow 160 in the flow cache, the routing device 140 is able to implement administrative policies which are designated for each message flow 160 rather than for each packet 150),

wherein different non-predefined sequences of components can be identified for different messages, each component being a software routine, and wherein dynamically identifying includes selecting individual components to create the non-predefined sequence of components (Kerr discloses a "plurality of components" for processing messages. For example, claim 1 of Kerr describes using a "plurality of devices" to apply "policy treatments" to a "plurality of messages," where policy treatments are used to perform "access control, ...security," "queuing," "accounting," "traffic profiling," etc. Id. at 10:27-40. Processing components can include "treatment with regard to switching," "access control," and "encryption." Id. at 4:20-34. "[S]pecial processing" can include "authentication" techniques "useful for implementing security 'firewalls.'" Id. at 35-46. Kerr further discloses that a "rewrite function" may be invoked "to alter the header for the packet." Id. at 4:55-62. These components can be used for "converting data with an input format into data with an output format," under Implicit' s apparent claim constructions, for example (as described above), the "encryption" and "rewrite" components to "alter" data to be processed. Id. at 4:30- 31, 4:55-62. The processing components of Kerr comprise "software routine" embodiments, as Kerr states that the processing instrumentality "may include specific hardware constructed or programmed performing the process steps described herein" or "a general purpose processor operating under program control." Id. at 2:51-55; see also id. at Figs. 3-4 (illustrating software data structures). Kerr discloses rather than applying a single predefined sequence to all flows, Kerr "determines proper treatment of packets 150 in the message fl0w" only when it "build a new entry in the flow cache" for that specific flow. Ex. 15 at 4:12-18. This includes, e.g., "determin[ing] encryption treatment for packets 150 in the message flow ... and any special treatment for packets 150 in the message flow." Id. at 4:31-34..); and

for each packet of each message, performing the processing of the identified non-predefined sequence of components of the message wherein state information generated by performing the processing of a component for a packet is available, to the component when the component processes the next packet of the message (After receiving the first packet of a new flow, Kerr builds a new flow entry that is cached in memory, which constitutes "storing". 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. 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. 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. 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).

In regard to claim 35, Kerr anticipates ***a computer-readable medium containing instructions*** (column 2, lines 30-32, However, those skilled in the art would recognize, after perusal of this application, that embodiments of the invention may be implemented using a set of general purpose computers operating under program control, and that modification of a set of general purpose computers to implement the process steps and data structures described herein would not require undue invention) ***for demultiplexing packets of messages*** (column 1 lines 59-60, The invention provides a method and system for switching in networks responsive to message flow patterns. A message "flow" is defined to comprise a set of packets to be transmitted between a particular source and a particular destination. When routers in a network identify a new message flow, they determine the proper processing for packets in that message flow and cache that information for that message flow. Thereafter, when routers in a network identify a packet which is part of that message flow, they process that packet according to the proper processing for packets in that message flow. The proper processing may include a determination of a destination port for routing those packets and a determination of whether access control permits routing those packets to their indicated destination), ***by method comprising:***

dynamically identifying a message-specific non-predefined sequence of components for processing the packets of each message upon receiving the first packet of the message wherein subsequent packets of the message can use the message-specific non-predefined sequence identified when the first packet was received (claim 1, lines 31-40, column 4 lines 20-34, column 3 line 38-column 6 line 27 identifying a first one message of a first plurality of messages associated with an application layer, said first plurality of messages having at least one policy treatment in common, said first plurality of messages being identified in response to an address of a selected source device and an address of a selected destination device, wherein said policy treatment comprises at least one of the access control information, security information, queuing information, accounting information, traffic profiling information, and policy information; In a preferred embodiment, the proper treatment of packets 150 in the message flow 160 includes treatment with regard to switching (thus, the routing device 140 determines an output port for switching packets 150 in the message flow 160), with regard to access control (thus, the routing device 140 determines whether packets 150 in the message flow 160 meet the requirements of access control, as defined by access control lists in force at the routing device 140), with regard to accounting (thus, the routing device 140 creates an accounting record for the message flow 160), with regard to encryption (thus, the routing device 140 determines encryption treatment for packets 150 in the message flow 160), and any special treatment for packets 150 in the message flow 160. FIG. 2 shows a method for routing in networks responsive to message flow patterns. In broad overview, the method for routing in networks responsive to message flow patterns comprises two parts. In a first part, the routing device 140 builds and uses a flow cache described in further detail with regard to FIG. 3), in which routing information to be used for packets 150 in each particular message flow 160 is recorded and from which such routing information is retrieved for use...A method 200 for routing in networks responsive to message flow patterns is performed by the routing device 140. At a flow point 210, the routing device 140 is disposed for building and using the flow cache. 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. At a step 223, the routing device 140 performs a lookup in a flow cache for the identified message flow 160. If the lookup is unsuccessful, the identified message flow 160 is a "new" message flow 160, and the routing device 140 continues with the step 224. If the lookup is successful, the identified message flow 160 is an "old" message flow 160, and the routing device 140 continues with the step 225. In a preferred embodiment, the routing device 140 determines a hash table key responsive to the flow key 310. This aspect of the step 223 is described in further detail with regard to FIG. 3. At a step 224, the routing device 140 builds a new entry in the flow cache. The routing device 140 determines proper treatment of packets 150 in the message flow 160 and enters information regarding such proper treatment in a data structure pointed to by the new entry in the flow cache. In a preferred embodiment, the routing device 140 determines the proper treatment by performing a lookup in an IP address cache as shown in FIG. 4. In a preferred embodiment, the proper treatment of packets 150 in the message flow 160 includes treatment with regard to switching (thus, the routing device 140 determines an output port for switching packets 150 in the message flow 160), with regard to access control (thus, the routing device 140 determines whether packets 150 in the message flow 160 meet the requirements of access control, as defined by access control lists in force at the routing device 140), with regard to accounting (thus, the routing device 140 creates an accounting record for the message flow 160), with regard to encryption (thus, the routing device 140 determines encryption treatment for packets 150 in the message flow 160), and any special treatment for packets 150 in the message flow 160. In a preferred embodiment, the routing device 140 performs any special processing for new message flows 160 at this time...Thereafter, the routing device 140 proceeds with the step 225, using the information from the new entry in the flow cache, just as if the identified message flow 160 were an "old" message flow 160 and the lookup in a flow cache had been successful. At a step 225, the routing device 140 retrieves routing information from the entry in the flow cache for the identified message flow 160. In a preferred embodiment, the entry in the flow cache includes a pointer to a rewrite function for at least part of a header for the packet 150. If this pointer is non-null, the routing device 140 invokes the rewrite function to alter the header for the packet 150. At a step 226, the routing device 140 routes the packet 150 responsive to the routing information retrieved at the step 225. 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. Thus, in a preferred embodiment, the routing device 140 routes the packet 150 to an output port, determines whether access is allowed for the packet 150, determines encryption treatment for the packet 150, and performs any special treatment for the packet 150, all responsive to information in the entry in the flow cache. In a preferred embodiment, the routing device 140 also enters accounting information in the entry in the flow cache for the packet 150. When routing each packet 150 in the message flow 160, the routing device 140 records the cumulative number of packets 150 and the cumulative number of bytes for the message flow 160. Because the routing device 140 processes each packet 150 in the message flow 160 responsive to the entry for the message flow 160 in the flow cache, the routing device 140 is able to implement administrative policies which are designated for each message flow 160 rather than for each packet 150),

and wherein dynamically identifying includes selecting individual components to create the message-specific non-predefined sequence of components (Kerr discloses a "plurality of components" for processing messages. For example, claim 1 of Kerr describes using a "plurality of devices" to apply "policy treatments" to a "plurality of messages," where policy treatments are used to perform "access control, ...security," "queuing," "accounting," "traffic profiling," etc. Id. at 10:27-40. Processing components can include "treatment with regard to switching," "access control," and "encryption." Id. at 4:20-34. "[S]pecial processing" can include "authentication" techniques "useful for implementing security 'firewalls.'" Id. at 35-46. Kerr further discloses that a "rewrite function" may be invoked "to alter the header for the packet." Id. at 4:55-62. These components can be used for "converting data with an input format into data with an output format," under Implicit' s apparent claim constructions, for example (as described above), the "encryption" and "rewrite" components to "'alter" data to be processed. Id. at 4:30- 31, 4:55-62. The processing components of Kerr comprise "software routine" embodiments, as Kerr states that the processing instrumentality "may include specific hardware constructed or programmed performing the process steps described herein" or "a general purpose processor operating under program control." Id. at 2:51-55; see also id. at Figs. 3-4 (illustrating software data structures)); and

for each packet of the message, invoking the identified non-predefined sequence of components in sequence to perform the processing of each component for the packet wherein each component saves message-specific state

information so that that component can use the saved message-specific state information when that component performs its processing on the next packet of the message (After receiving the first packet of a new flow, Kerr builds a new flow entry that is cached in memory, which constitutes "storing". 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. 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. 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. 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).

11. Claims 15 and 35 are rejected under 35 U.S.C. 102(b) as being anticipated by Decasper98.

In regards to claim 15, Decasper98 anticipates **a method in a computer system**

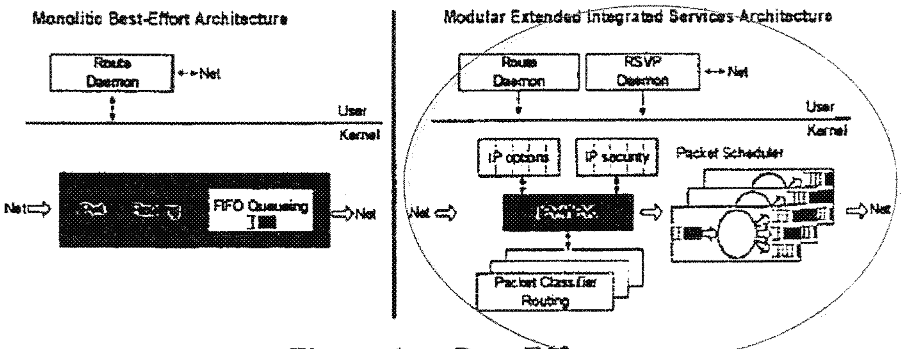


Figure 1. : Best Effort vs
Extended Integrated Services Router (EISR)

for demultiplexing packets of messages (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 0fmultimedia data sources and applications (e.g. real-time audio/video) will produce longer lived packet streams with more packets per session than is common in today's environment, an integrated services router architecture should support the notion of flows and build upon it. Id. at 3. A flow is defined as a group of packets which satisfy a specific filter. See id. at 3 ("Sets of flows are specified using filters .... Filters can also match individual end-to-end application flows"). Id. at 3. A flow would comprise a "message" under Implicit's apparent claim constructions. See section IV.C), ***the method comprising:***

dynamically identifying a non-predefined sequence of components for processing each message based on the first packet of the message so that subsequent packets of the message can be processed without re-identifying the components, wherein different non-predefined sequences of components can be

identified for different messages (when the first packet of a new flow arrives, Decasper98 performs an expensive series of filter operations to determine the correct sequence of plugin components to be applied to the flow. See Ex. 25 at 5-6 ("The processing of the first packet of a new flow.., involves n filter table lookups to create a single entry in the flow table for the new flow."). This expensive series of filter operations does not need to be repeated for subsequent packets of the flow, because the new "entry... in the flow" table serves as a fast cache for future lookup of packets belonging to that flow," and the entry "stores pointers to the appropriate plugins." Id. at 5. Performance is thus enhanced for subsequent packets of the flow, since "[u]sually, filter table lookups are much slower than flow table lookups." Id. See also id. at 3 ("Subsequent packets get this information from a fast flow cache which temporarily stores the information gathered by processing the first packet." Decasper98 explains: "it is very important to be able to... apply different policies to different flows." Ex. 25 at 2. This is why Decasper98 applies a series of filters to each flow, wherein each filter may select a specific plugin component implementing a different policy. See Id. at 5-7).

Decasper98 assigns the sequence of plugins to the flow on the basis of lookups in multiple independent "filter tables." E.g., id. at 5-7 ("The processing of the first packet of a new flow.., involves n filter table lookup to create a single entry in the flow table for the new flow"); 7 ("multiple lookups (in different filter tables)"). E.g., a first filter table determines whether a first plugin is added to the sequence, a second independent filter table determines whether a second plugin is added, a third independent filter table determines whether a third plugin is added, and so on. See id. at 5-7.

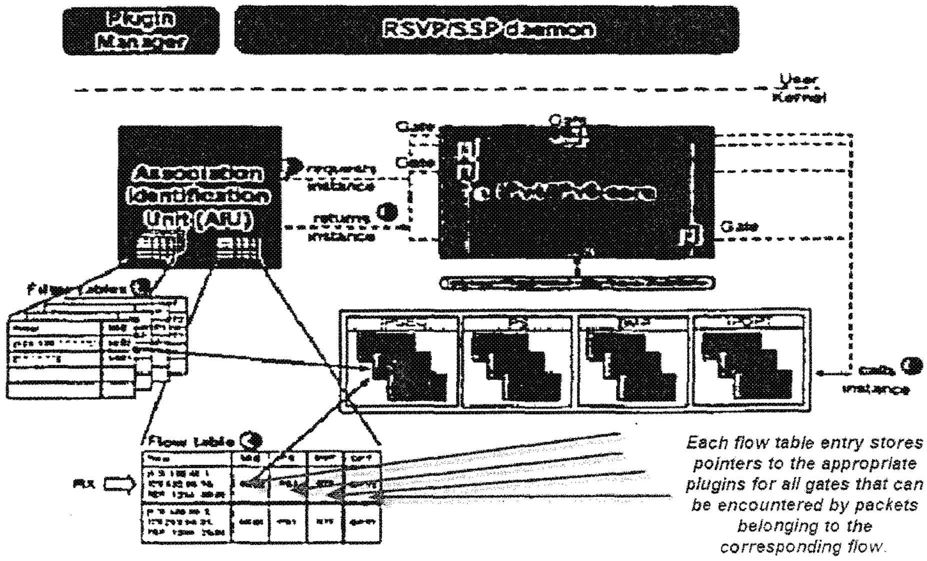
This leads "exponentially" to an enormous number of possible sequences that might be applied to the first packet of a flow when it arrives, "even with very few installed filters." See Id. at 7.2. These various possible sequences are not stored or enumerated anywhere in the system ahead of time. Instead, the sequence of plugins for a flow is generated algorithmically when the first packet of a flow arrives, by applying a series of filter operation to packet data which was not available to the system until that moment. See id. at 5-7.

Decasper98 explicitly considers and rejects a "theoretically possible" alternative approach, which is to replace this system of multiple independent filters with "a single global filter table." Id. at 7. Under this alternative approach, only a single filter would apply to a particular flow, and that single filter would specify the entire sequence of components to be applied to it. See Id. When the first packet arrived, the system would find the single matching filter and then essentially just read off the sequence of components to be applied to that flow. See Id. Thus, the sequence would be pre-defined and readily identifiable as such in a specific filter entry, even before the first packet arrived.

However, Decasper98 rejects this approach as "practically infeasible because the space requirements for the global table can, even with very few installed filters, increase very quickly (exponentially) to unacceptable levels." Id. In other words, Decasper98's multiple filter table approach implies so many potential valid sequences that it is impossible to even enumerate them all ahead of time in memory--since they would not fit.

Instead, Decasper98 adopts an algorithmic approach where the correct sequence is generated dynamically on demand, by applying the series of multiple filters to the first packet when it arrives. Thus, under Implicit's apparent claim constructions, Decasper98 discloses "for the first packet of the message, dynamically identifying a non-predefined sequence of components for processing the packets of the message."

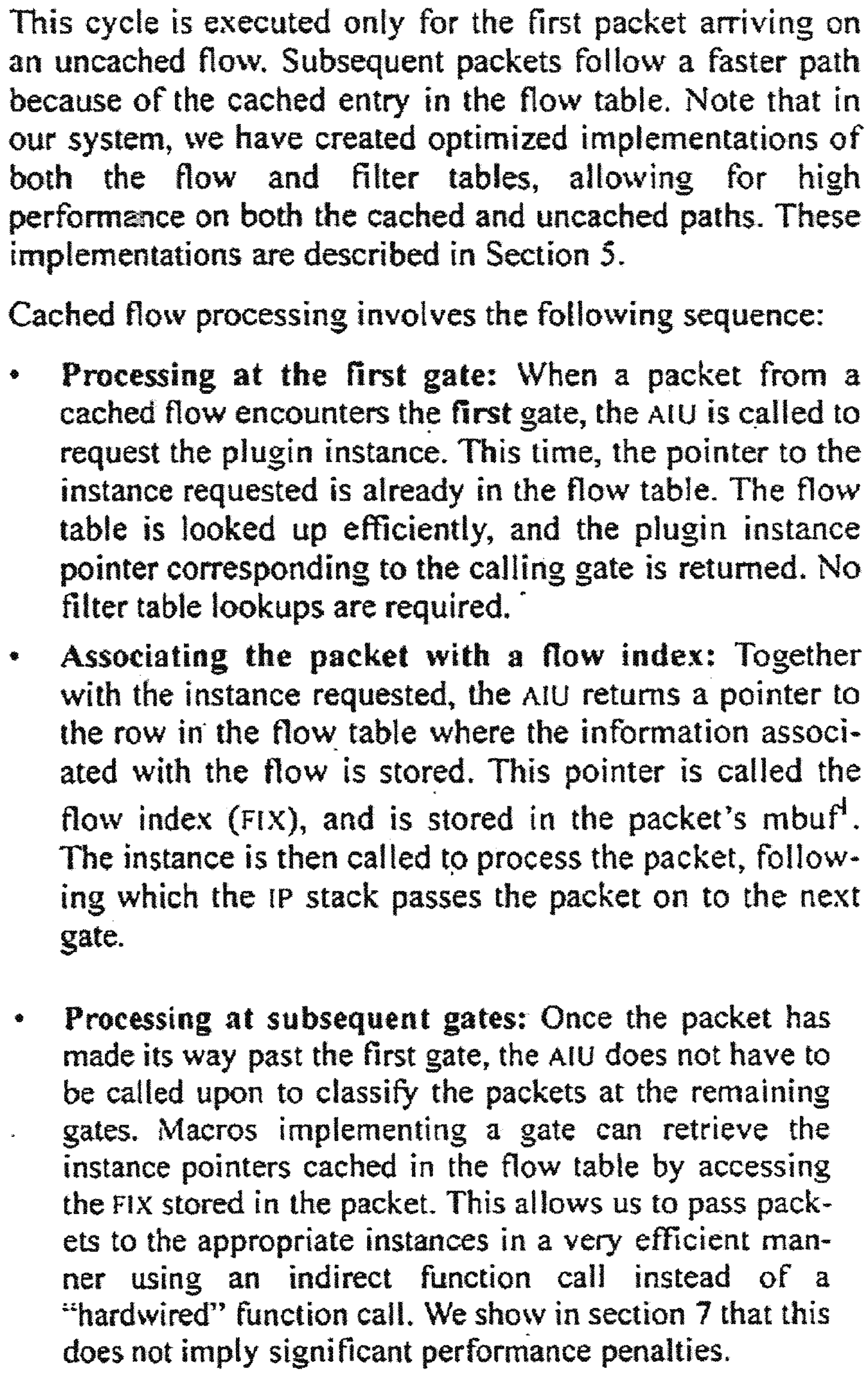
Decasper98 discloses this "dynamically identifying" claim element under Implicit's apparent claim constructions. Decasper98 also teaches "selecting individual components to create the non-predefined sequence of components after the first packet is received." As explained above, after the first packet of a flow arrives, Descapser98 applies a series of independent filters to it, each of which may select a different individual plugin. Id at 5-7. See also, e.g., id. at 4 (Figure 2, showing various individual plugins that might be selected within each category, e.g., "BMP1 BMP2 BMP3"). The very purpose of this architecture is to apply the fight specific individual plugins in a tailored manner to each particular flow. E.g., id. at 2 ("it is very important to be able to quickly and efficiently classify packets into flows, and to apply different policies to different flows"), 3, 7.



Each flow table entry stores pointers to the appropriate plugins for all gates that can be encountered by packets belonging to the corresponding flow.

Figure 3. : System Architecture and Data Path

In the AIU, all flows start out being uncached (i.e., they do not have an entry in the flow table). If an incoming packet belongs to an uncached flow, its lookup in the flow table data structure will fail (i.e., there is a cache miss). In this case, the packet needs to be looked up in a different data structure that we call a **filter table**. Filter tables store the bindings between filters and plugins for each gate. The filter table lookup algorithm finds the most specific matching filter (described later) that has been installed in the table, and returns the corresponding plugin instance. 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. The processing of the first packet of a new flow with *n* gates involves *n* filter table lookups to create a single entry in the flow table for the new flow.



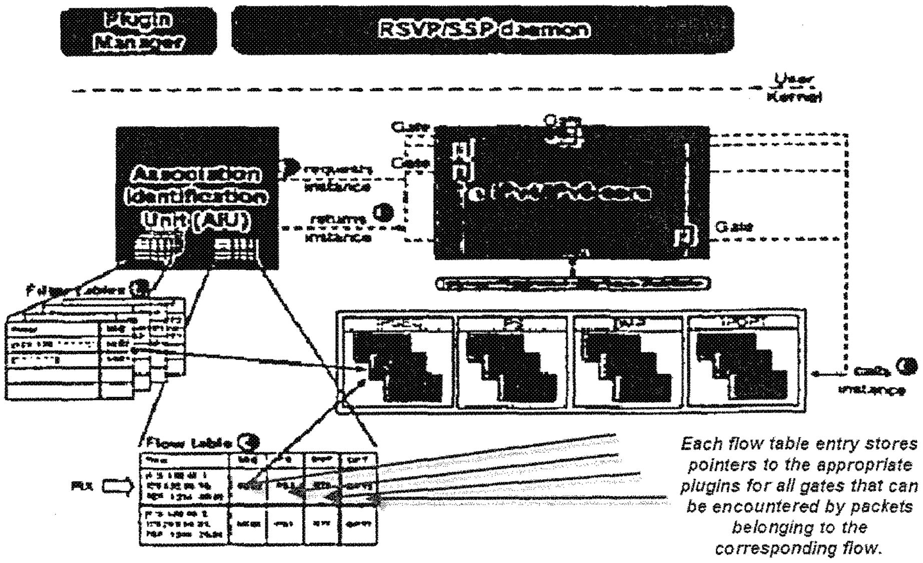
This cycle is executed only for the first packet arriving on an uncached flow. Subsequent packets follow a faster path because of the cached entry in the flow table. Note that in our system, we have created optimized implementations of both the flow and filter tables, allowing for high performance on both the cached and uncached paths. These implementations are described in Section 5.

Cached flow processing involves the following sequence:

- **Processing at the first gate:** When a packet from a cached flow encounters the **first** gate, the AIU is called to request the plugin instance. This time, the pointer to the instance requested is already in the flow table. The flow table is looked up efficiently, and the plugin instance pointer corresponding to the calling gate is returned. No filter table lookups are required.
- **Associating the packet with a flow index:** Together with the instance requested, the AIU returns a pointer to the row in the flow table where the information associated with the flow is stored. This pointer is called the flow index (FIX), and is stored in the packet's mbuf[1]. The instance is then called to process the packet, following which the IP stack passes the packet on to the next gate.
- **Processing at subsequent gates:** Once the packet has made its way past the first gate, the AIU does not have to be called upon to classify the packets at the remaining gates. Macros implementing a gate can retrieve the instance pointers cached in the flow table by accessing the FIX stored in the packet. This allows us to pass packets to the appropriate instances in a very efficient manner using an indirect function call instead of a "hardwired" function call. We show in section 7 that this does not imply significant performance penalties.

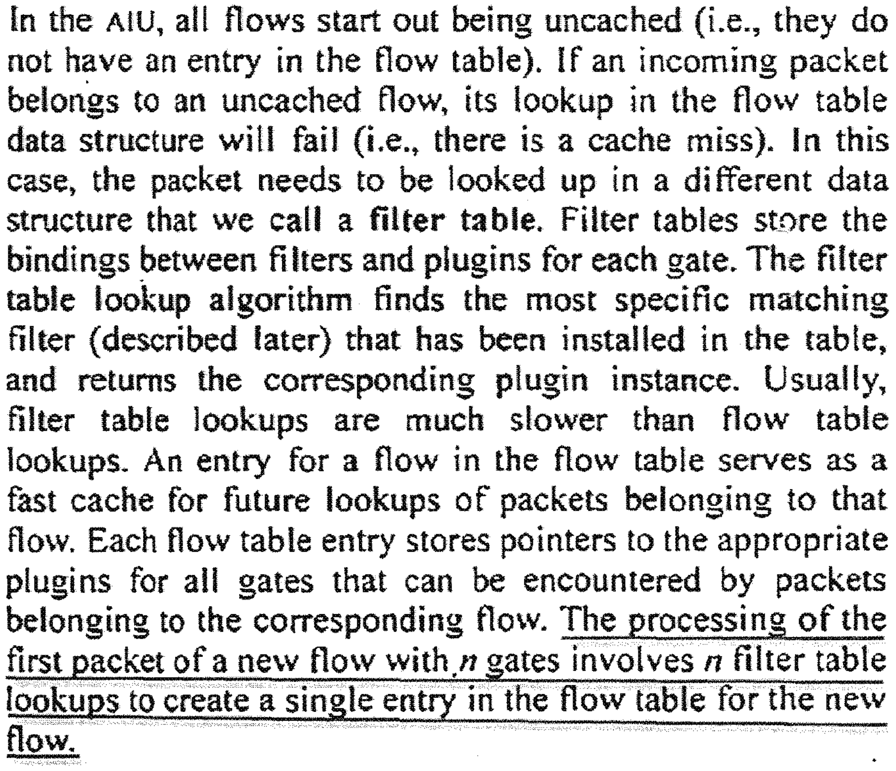
each component being a software routine (Id. at 2 ("plugins are kernel software modules that are.., responsible for performing certain functions on specified network flows.") for the first packet of the message, dynamically identifying a non-predefined sequence of components for processing the packets of the message wherein dynamically identifying includes selecting individual components to create the non-predefined sequence of components after the first packet is received (When the first packet of a new flow arrives, Decasper98 performs an expensive series of filter operations to determine the correct sequence of plugin components to be applied to the flow. See Ex. 25 at 5-6 ("The processing of the first packet of a new flow.., involves n filter table lookups to create a single entry in the flow table for the new flow."). This expensive series of filter operations does not need to be repeated for subsequent packets of the flow, because the new "entry... in the flow" table serves as a fast cache for future lookup of packets belonging to that flow," and the entry "stores pointers to the appropriate plugins." Id. at 5. Performance is thus enhanced for subsequent packets of the flow, since "[u]sually, filter table lookups are much slower than flow table lookups." Id. See also id. at 3 ("Subsequent packets get this information from a fast flow cache which temporarily stores the information gathered by processing the first packet."). Decasper98 assigns the sequence of plugins to the flow on the basis of lookups in multiple independent "filter tables." E.g., id. at 5-7 ("The processing of the first packet of a new flow.., involves n filter table lookup to create a single entry in the flow table for the new flow"); 7 ("multiple lookups (in different filter tables)"). E.g., a first filter table determines whether a first plugin is added to the sequence, a second independent filter table determines whether a second plugin is added, a third independent filter table determines whether a third plugin is added, and so on. See Id. at 5-7, and wherein dynamically identifying includes selecting individual components to create the non-predefined sequence of components; and

for each packet of each message, performing the processing of the identified non-predefined sequence of components of the message wherein state information generated by performing the processing of a component for a packet is available, to the component when the component processes the next packet of the message (

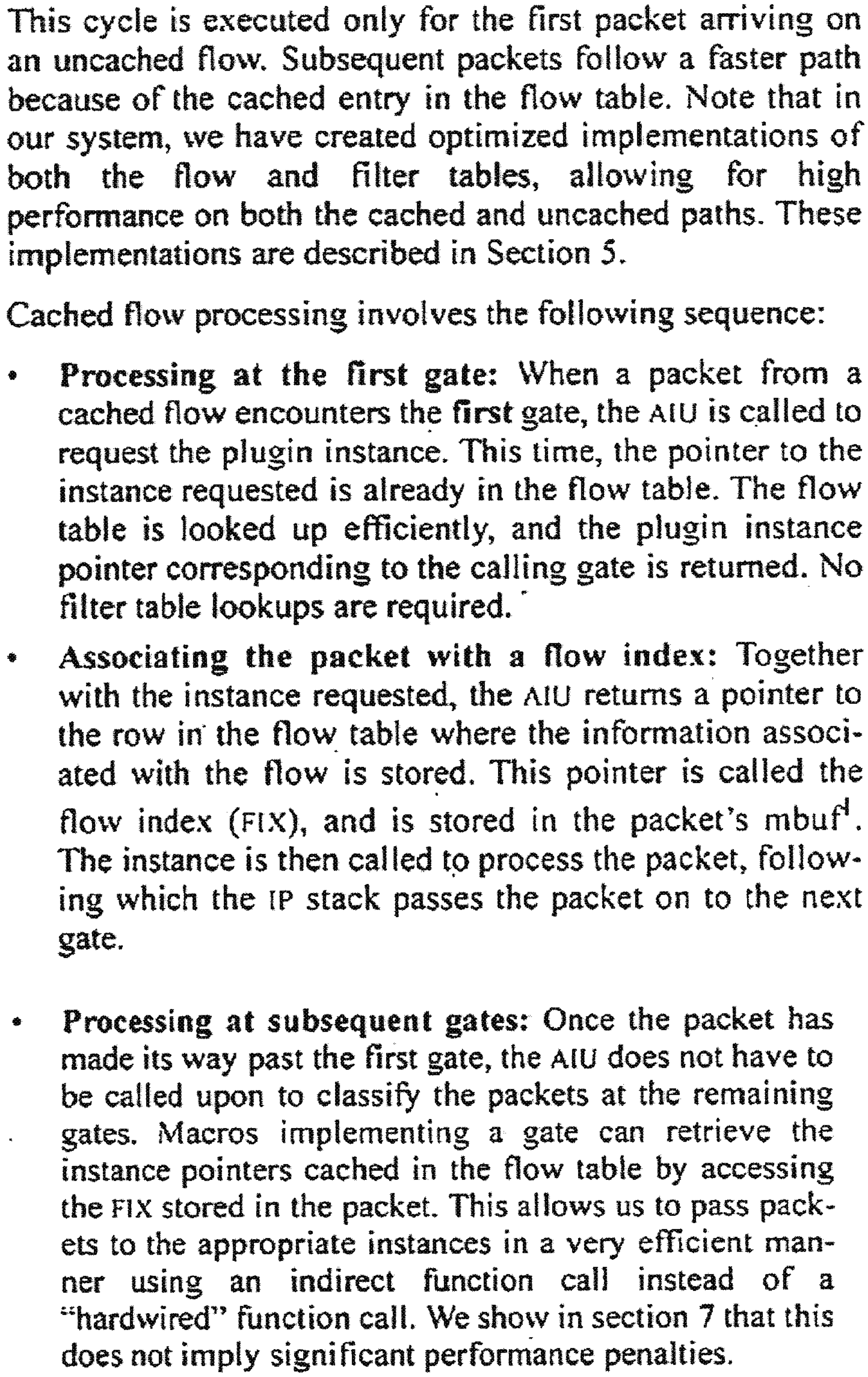


Each flow table entry stores pointers to the appropriate plugins for all gates that can be encountered by packets belonging to the corresponding flow.

Figure 3. : System Architecture and Data Path



In the AIU, all flows start out being uncached (i.e., they do not have an entry in the flow table). If an incoming packet belongs to an uncached flow, its lookup in the flow table data structure will fail (i.e., there is a cache miss). In this case, the packet needs to be looked up in a different data structure that we call a **filter table**. Filter tables store the bindings between filters and plugins for each gate. The filter table lookup algorithm finds the most specific matching filter (described later) that has been installed in the table, and returns the corresponding plugin instance. 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. The processing of the first packet of a new flow with *n* gates involves *n* filter table lookups to create a single entry in the flow table for the new flow.



This cycle is executed only for the first packet arriving on an uncached flow. Subsequent packets follow a faster path because of the cached entry in the flow table. Note that in our system, we have created optimized implementations of both the flow and filter tables, allowing for high performance on both the cached and uncached paths. These implementations are described in Section 5.

Cached flow processing involves the following sequence:

- **Processing at the first gate:** When a packet from a cached flow encounters the **first** gate, the AIU is called to request the plugin instance. This time, the pointer to the instance requested is already in the flow table. The flow table is looked up efficiently, and the plugin instance pointer corresponding to the calling gate is returned. No filter table lookups are required.
- **Associating the packet with a flow index:** Together with the instance requested, the AIU returns a pointer to the row in the flow table where the information associated with the flow is stored. This pointer is called the flow index (FIX), and is stored in the packet's mbuf[1]. The instance is then called to process the packet, following which the IP stack passes the packet on to the next gate.

- **Processing at subsequent gates:** Once the packet has made its way past the first gate, the AIU does not have to be called upon to classify the packets at the remaining gates. Macros implementing a gate can retrieve the instance pointers cached in the flow table by accessing the FIX stored in the packet. This allows us to pass packets to the appropriate instances in a very efficient manner using an indirect function call instead of a "hardwired" function call. We show in section 7 that this does not imply significant performance penalties.

).

In regards to claim 35, Decasper98 anticipates ***a computer-readable medium containing instructions***

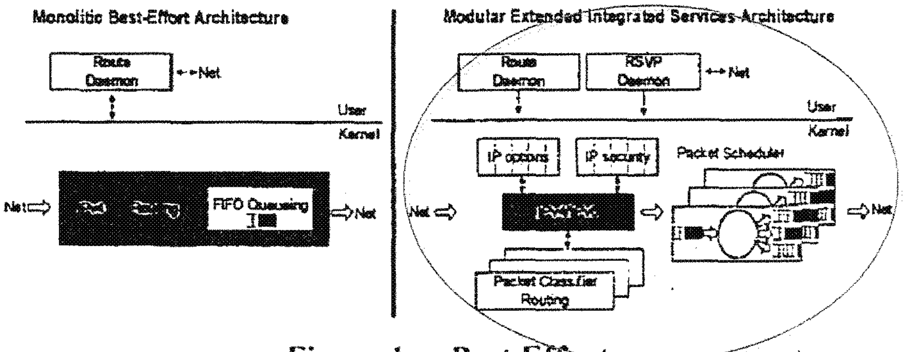


**Figure 1. : Best Effort vs
Extended Integrated Services Router (EISR)**

for demultiplexing packets of messages (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 0fmultimedia data sources and applications (e.g. real-time audio/video) will produce longer lived packet streams with more packets per session than is common in today's environment, an integrated services router architecture should support the notion of flows and build upon it. Id. at 3. A flow is defined as a group of packets which satisfy a specific filter. See id. at 3 ("Sets of flows are specified using filters .... Filters can also match individual end-to-end application flows"). Id. at 3. A flow would comprise a "message" under Implicit's apparent claim constructions. See section IV.C), ***by method comprising***:

dynamically identifying a message-specific non-predefined sequence of components for processing the packets of each message upon receiving the first

packet of the message wherein subsequent packets of the message can use the message-specific non-predefined sequence identified when the first packet was received, and wherein dynamically identifying includes selecting individual components to create the message-specific non-predefined sequence of components (when the first packet of a new flow arrives, Decasper98 performs an expensive series of filter operations to determine the correct sequence of plugin components to be applied to the flow. See Ex. 25 at 5-6 ("The processing of the first packet of a new flow.., involves n filter table lookups to create a single entry in the flow table for the new flow."). This expensive series of filter operations does not need to be repeated for subsequent packets of the flow, because the new "entry... in the flow" table serves as a fast cache for future lookup of packets belonging to that flow," and the entry "stores pointers to the appropriate plugins." Id. at 5. Performance is thus enhanced for subsequent packets of the flow, since "[u]sually, filter table lookups are much slower than flow table lookups." Id. See also id. at 3 ("Subsequent packets get this information from a fast flow cache which temporarily stores the information gathered by processing the first packet.").

Decasper98 assigns the sequence of plugins to the flow on the basis of lookups in multiple independent "filter tables." E.g., id. at 5-7 ("The processing of the first packet of a new flow.., involves n filter table lookup to create a single entry in the flow table for the new flow"); 7 ("multiple lookups (in different filter tables)"). E.g., a first filter table determines whether a first plugin is added to the sequence, a second independent filter

table determines whether a second plugin is added, a third independent filter table determines whether a third plugin is added, and so on. See id. at 5-7.

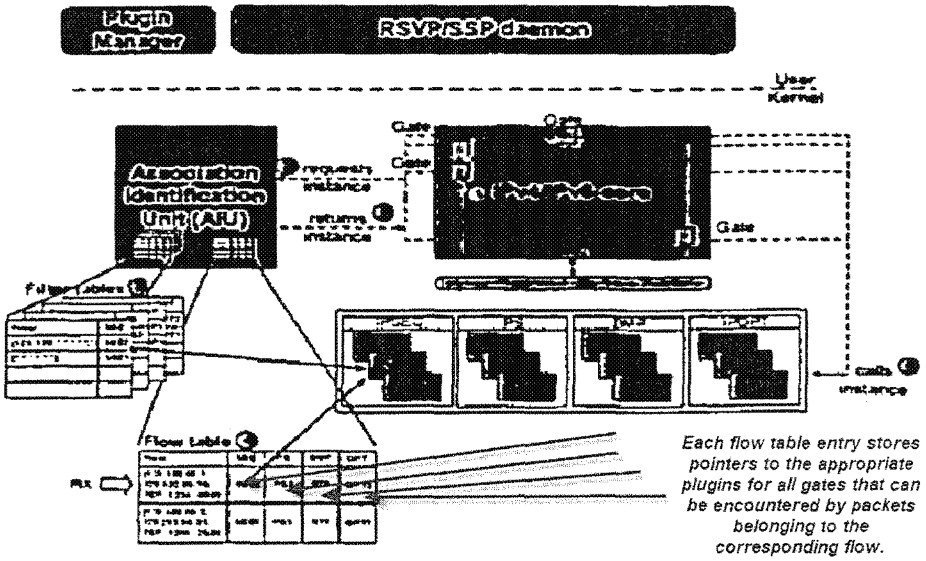
This leads "exponentially" to an enormous number of possible sequences that might be applied to the first packet of a flow when it arrives, "even with very few installed filters." See Id. at 7.2. These various possible sequences are not stored or enumerated anywhere in the system ahead of time. Instead, the sequence of plugins for a flow is generated algorithmically when the first packet of a flow arrives, by applying a series of filter operation to packet data which was not available to the system until that moment. See id. at 5-7.

Decasper98 explicitly considers and rejects a "theoretically possible" alternative approach, which is to replace this system of multiple independent filters with "a single global filter table." Id. at 7. Under this alternative approach, only a single filter would apply to a particular flow, and that single filter would specify the entire sequence of components to be applied to it. See Id. When the first packet arrived, the system would find the single matching filter and then essentially just read off the sequence of components to be applied to that flow. See Id. Thus, the sequence would be pre-defined and readily identifiable as such in a specific filter entry, even before the first packet arrived.

However, Decasper98 rejects this approach as "practically infeasible because the space requirements for the global table can, even with very few installed filters, increase very quickly (exponentially) to unacceptable levels." Id. In other words, Decasper98's multiple filter table approach implies so many potential valid sequences that it is impossible to even enumerate them all ahead of time in memory--since they would not fit.

Instead, Decasper98 adopts an algorithmic approach where the correct sequence is generated dynamically on demand, by applying the series of multiple filters to the first packet when it arrives. Thus, under Implicit's apparent claim constructions, Decasper98 discloses "for the first packet of the message, dynamically identifying a non-predefined sequence of components for processing the packets of the message."

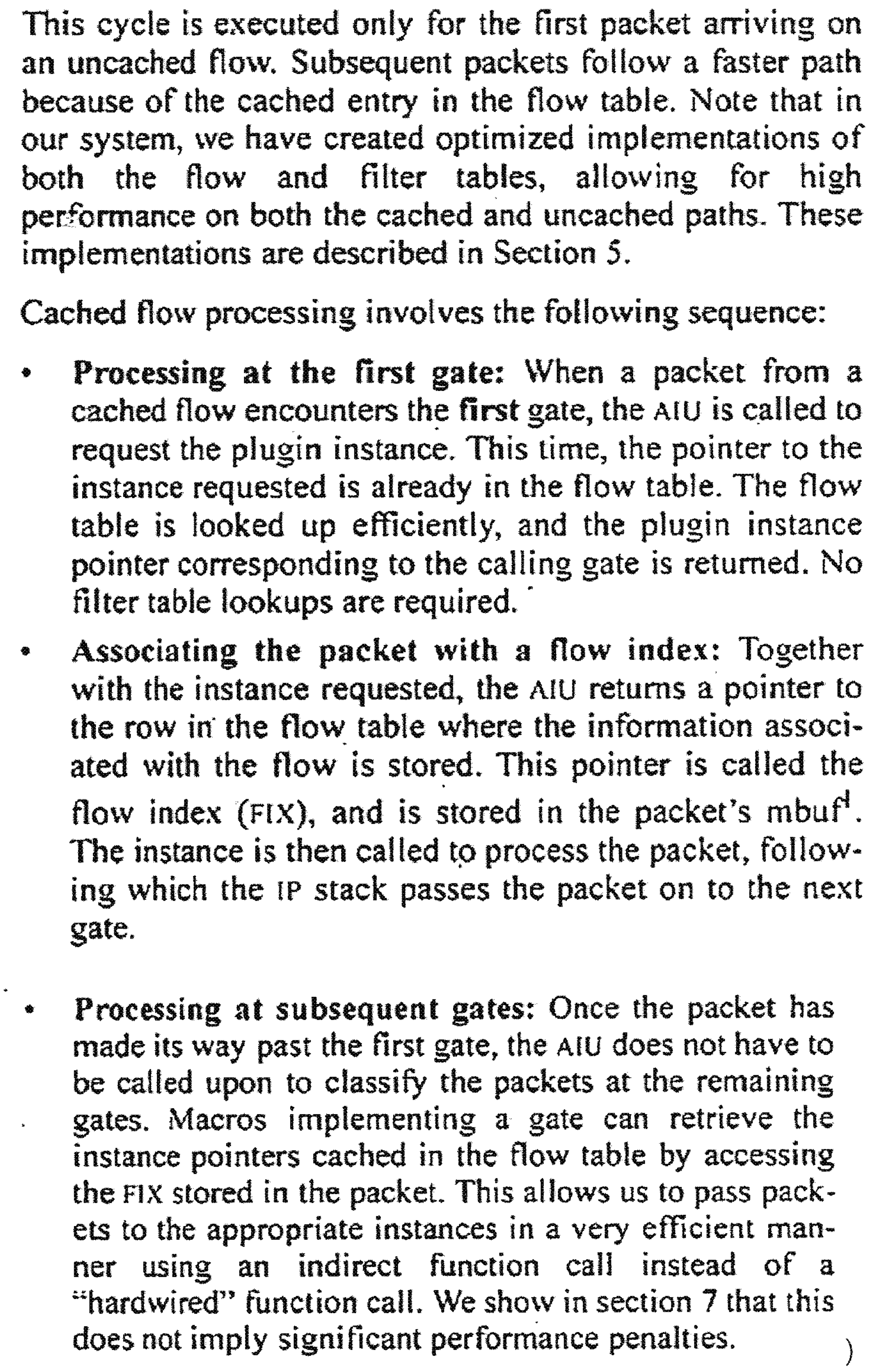
Decasper98 discloses this "dynamically identifying" claim element under Implicit's apparent claim constructions. Decasper98 also teaches "selecting individual components to create the non-predefined sequence of components after the first packet is received." As explained above, after the first packet of a flow arrives, Descapser98 applies a series of independent filters to it, each of which may select a different individual plugin. Id at 5-7. See also, e.g., id. at 4 (Figure 2, showing various individual plugins that might be selected within each category, e.g., "BMP1 BMP2 BMP3"). The very purpose of this architecture is to apply the fight specific individual plugins in a tailored manner to each particular flow. E.g., id. at 2 ("it is very important to be able to quickly and efficiently classify packets into flows, and to apply different policies to different flows"), 3, 7.



Each flow table entry stores pointers to the appropriate plugins for all gates that can be encountered by packets belonging to the corresponding flow.

Figure 3. : System Architecture and Data Path

In the AIU, all flows start out being uncached (i.e., they do not have an entry in the flow table). If an incoming packet belongs to an uncached flow, its lookup in the flow table data structure will fail (i.e., there is a cache miss). In this case, the packet needs to be looked up in a different data structure that we call a **filter table**. Filter tables store the bindings between filters and plugins for each gate. The filter table lookup algorithm finds the most specific matching filter (described later) that has been installed in the table, and returns the corresponding plugin instance. 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. The processing of the first packet of a new flow with *n* gates involves *n* filter table lookups to create a single entry in the flow table for the new flow.



This cycle is executed only for the first packet arriving on an uncached flow. Subsequent packets follow a faster path because of the cached entry in the flow table. Note that in our system, we have created optimized implementations of both the flow and filter tables, allowing for high performance on both the cached and uncached paths. These implementations are described in Section 5.

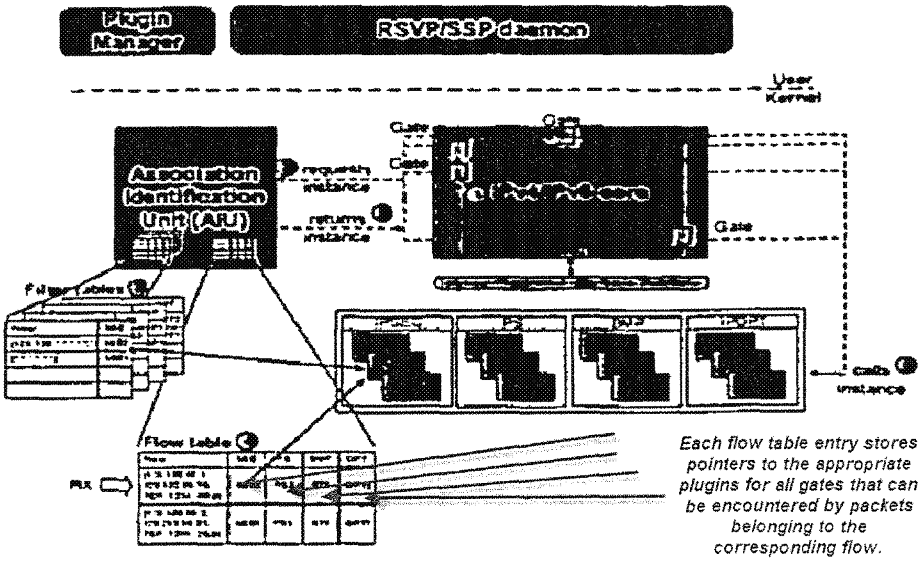
Cached flow processing involves the following sequence:

- **Processing at the first gate:** When a packet from a cached flow encounters the **first** gate, the AIU is called to request the plugin instance. This time, the pointer to the instance requested is already in the flow table. The flow table is looked up efficiently, and the plugin instance pointer corresponding to the calling gate is returned. No filter table lookups are required.
- **Associating the packet with a flow index:** Together with the instance requested, the AIU returns a pointer to the row in the flow table where the information associated with the flow is stored. This pointer is called the flow index (FIX), and is stored in the packet's mbuf[1]. The instance is then called to process the packet, following which the IP stack passes the packet on to the next gate.
- **Processing at subsequent gates:** Once the packet has made its way past the first gate, the AIU does not have to be called upon to classify the packets at the remaining gates. Macros implementing a gate can retrieve the instance pointers cached in the flow table by accessing the FIX stored in the packet. This allows us to pass packets to the appropriate instances in a very efficient manner using an indirect function call instead of a "hardwired" function call. We show in section 7 that this does not imply significant performance penalties.

)

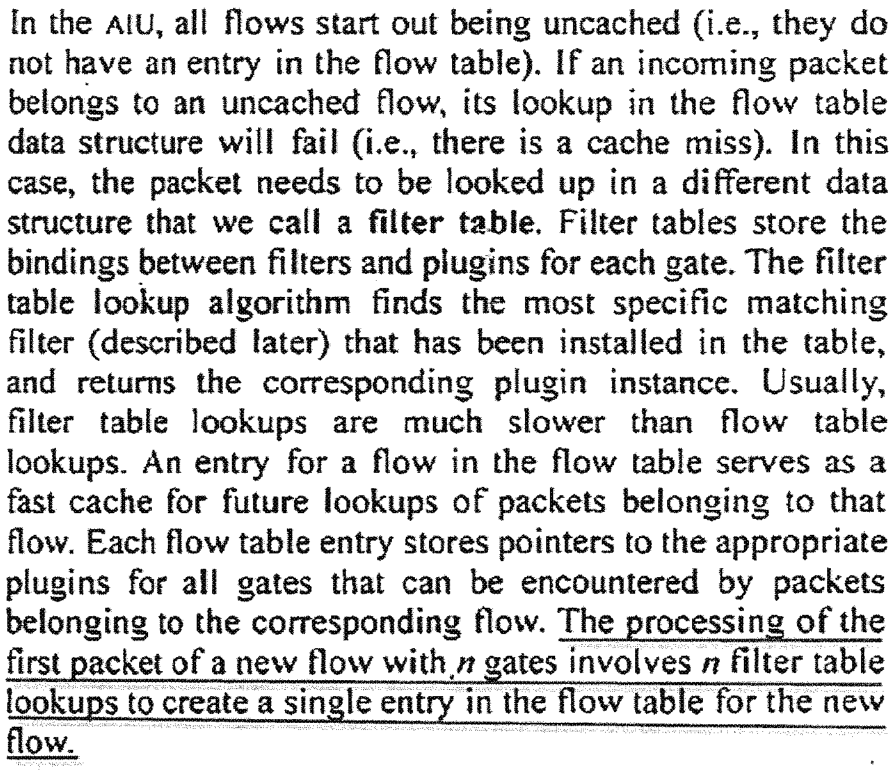
and for each packet of the message, invoking the identified non-predefined sequence of components in sequence to perform the processing of each

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

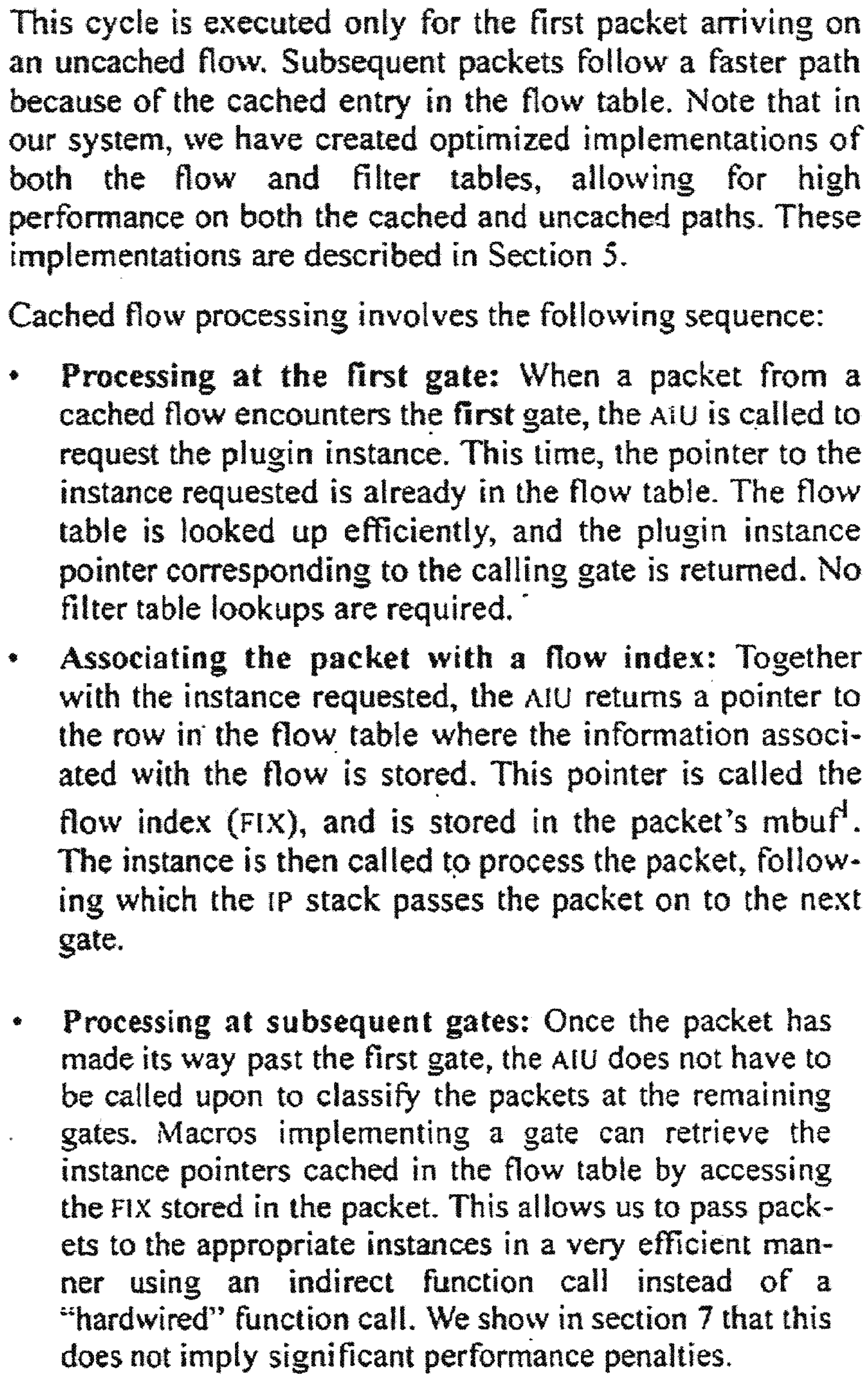


Each flow table entry stores pointers to the appropriate plugins for all gates that can be encountered by packets belonging to the corresponding flow.

Figure 3. : System Architecture and Data Path



In the AIU, all flows start out being uncached (i.e., they do not have an entry in the flow table). If an incoming packet belongs to an uncached flow, its lookup in the flow table data structure will fail (i.e., there is a cache miss). In this case, the packet needs to be looked up in a different data structure that we call a **filter table**. Filter tables store the bindings between filters and plugins for each gate. The filter table lookup algorithm finds the most specific matching filter (described later) that has been installed in the table, and returns the corresponding plugin instance. 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. The processing of the first packet of a new flow with *n* gates involves *n* filter table lookups to create a single entry in the flow table for the new flow.



This cycle is executed only for the first packet arriving on an uncached flow. Subsequent packets follow a faster path because of the cached entry in the flow table. Note that in our system, we have created optimized implementations of both the flow and filter tables, allowing for high performance on both the cached and uncached paths. These implementations are described in Section 5.

Cached flow processing involves the following sequence:

- **Processing at the first gate:** When a packet from a cached flow encounters the **first** gate, the AIU is called to request the plugin instance. This time, the pointer to the instance requested is already in the flow table. The flow table is looked up efficiently, and the plugin instance pointer corresponding to the calling gate is returned. No filter table lookups are required.
- **Associating the packet with a flow index:** Together with the instance requested, the AIU returns a pointer to the row in the flow table where the information associated with the flow is stored. This pointer is called the flow index (FIX), and is stored in the packet's mbuf[1]. The instance is then called to process the packet, following which the IP stack passes the packet on to the next gate.
- **Processing at subsequent gates:** Once the packet has made its way past the first gate, the AIU does not have to be called upon to classify the packets at the remaining gates. Macros implementing a gate can retrieve the instance pointers cached in the flow table by accessing the FIX stored in the packet. This allows us to pass packets to the appropriate instances in a very efficient manner using an indirect function call instead of a "hardwired" function call. We show in section 7 that this does not imply significant performance penalties.

).

Service of Papers

12. After the filing of a request for reexamination by a third party requester, any document filed by either the patent owner or the third party requester must be served on the other party (or parties where two or more third party requester proceedings are merged) in the reexamination proceeding in the manner provided in 37 CFR 1.248. See 37 CFR 1.550(t').

Extensions of Time

13. Extensions of time under 37 CFR 1.136(a) will not be permitted in inter partes reexamination proceedings because the provisions of 37 CFR 1.136 apply only to "an applicant" and not to parties in a reexamination proceeding. Additionally, 35 U.S.C. 314(c) requires that inter partes reexamination proceedings "will be conducted with special dispatch" (37 CFR 1.937). Patent owner extensions of time in inter partes reexamination proceedings are provided for in 37 CFR 1.956. Extensions of time are not available for third party requester comments, because a comment period of 30 days from service of patent owner's response is set by statute 35 U.S.C. 314(b)(3). Time periods may be extended only upon a strong showing of sufficient cause.

Notification of Concurrent Proceedings

14. The patent owner is reminded of the continuing responsibility under 37 CFR 1.985(a), to apprise the Office of any litigation activity, or other prior or concurrent proceeding, involving the 6,629,163 patent throughout the course of this reexamination

proceeding. The third party requester is also reminded of the ability to similarly apprise the Office of any such activity or proceeding throughout the course of this reexamination proceeding. See MPEP 2686 and 2686.04.

Conclusion

15. **This is an ACTION CLOSING PROSECUTION (ACP)**; see MPEP § 2671.02.

16. (1) Pursuant to 37 CFR 1.951(a), the patent owner may once file written comments limited to the issues raised in the reexamination proceeding and/or present a proposed amendment to the claims which amendment will be subject to the criteria of 37 CFR 1.116 as to whether it shall be entered and considered. Such comments and/or proposed amendments must be filed within a time period of 30 days or one month (whichever is longer) from the mailing date of this action. Where the patent owner files such comments and/or a proposed amendment, the third party requester may once file comments under 37 CFR 1.951(b) responding to the patent owner's submission within 30 days from the date of service of the patent owner's submission on the third party requester.

17. (2) If the patent owner does not timely file comments and/or a proposed amendment pursuant to 37 CFR 1.951(a), then the third party requester is precluded from filing comments under 37 CFR 1.951(b).

18. (3) Appeal **cannot** be taken from this action, since it is not a final Office action.

19. All correspondence relating to this *inter partes* reexamination proceeding should be directed:

By EFS: Registered users may submit via the electronic filing system EFS-Web, at https://efs.uspto.gov/efile/myportal/efs-registered

By Mail to: *Mail Stop Inter Partes* Reexam
Attn: Central Reexamination Unit
Commissioner for Patents
United States Patent & Trademark Office
P.O. Box 1450
Alexandria, Virginia 22313-1450

By FAX to: (571) 273-9900
Central Reexamination Unit

By hand: Customer Service Window
Attn: Central Reexamination Unit
Randolph Building, Lobby Level
401 Dulany Street
Alexandria, VA 22314

For EFS-Web transmissions, 37 CFR 1.8(a)(1)(i) (C) and (ii) states that correspondence (except for a request for reexamination and a corrected or replacement request for reexamination) will be considered timely filed if (a) it is transmitted via the Office's electronic filing system in accordance with 37 CFR 1.6(a)(4), and (b) includes a certificate of transmission for each piece of correspondence stating the data of transmission, which is prior to the expiration of the set period of time in the Office action.

Any inquiry concerning this communication or earlier communications from the examiner, or as to the status of this proceeding, should be directed to the Central Reexamination Unit at telephone number (571) 272-7705.

/Salman Ahmed/
Salman Ahmed
Primary Examiner
Central Reexamination Unit - Art Unit 3992
(571) 272-8307

Conferee: /Ovidio Escalante/

Conferee: /Daniel Ryman/

Reexamination	Application/Control No. 95000659	Applicant(s)/Patent Under Reexamination 6629163
	Certificate Date	Certificate Number

Requester Correspondence Address: □ **Patent Owner** ☒ **Third Party**

Irell and Manella, LLP
830 Newport Center Dr. , Ste 400
Newport Beach CA 92660

LITIGATION REVIEW ☒	/SA/ (examiner initials)	03/28/2012 (date)

Case Name	Director Initials
3:10cv4234 (OPEN) Implicit Networks, Inc v. Juniper Networks,	DJR fr IY
3:10cv3766 (CLOSED) Implicit Networks, Inc v. Citrix Systems,	
3:10cv3746 (OPEN) Implicit Networks, Inc v. Hewlett-Packard	
5:10cv3606 (OPEN) Implicit Networks, Inc v. Cisco Systems,	
3:10cv3606 (CLOSED) Implicit Networks, Inc v. Cisco Systems	
3:10cv3365 (OPEN) Implicit Networks, Inc v. F5 Networks	
3:09cv5628 (CLOSED) Implicit Networks, Inc v. Microsoft	
2:08cv184(CLOSED) Implicit Networks, Inc v. A M Devices	

COPENDING OFFICE PROCEEDINGS

TYPE OF PROCEEDING	NUMBER

Implicit Exhibit 2003
Juniper v. Implicit

Reexamination	Application/Control No. 95000659	Applicant(s)/Patent Under Reexamination 6629163
	Certificate Date	Certificate Number

COPENDING OFFICE PROCEEDINGS	
TYPE OF PROCEEDING	NUMBER

Search Notes	Application/Control No. 95000659	Applicant(s)/Patent Under Reexamination 6629163
	Examiner SALMAN AHMED	Art Unit 3992

SEARCHED

Class	Subclass	Date	Examiner
None	None		

SEARCH NOTES

Search Notes	Date	Examiner
File hstory	3/28/2012	SA
Patent prosecution history	3/28/2012	SA

INTERFERENCE SEARCH

Class	Subclass	Date	Examiner

Organization CRU Bldg./Room MDG 5D20
UNITED STATES PATENT AND TRADEMARK OFFICE
P.O. BOX 1450
ALEXANDRIA, VA 22313-1450
IF UNDELIVERABLE RETURN IN TEN DAYS

OFFICIAL BUSINESS
Penalty for Private Use, $300

AN EQUAL OPPORTUNITY EMPLOYER

Priority Mail
ComBasPrice

UNITED STATES POSTAGE
PITNEY BOWES
$ 09.62⁰
[illegible] OCT 02 2012
MAILED FROM ZIPCODE 22314

IRELL & MANELLA, LLP
DAVID MCPHIE
840 NEWPORT CENTER DR., STE 400
NEWPORT BEACH, CA 92660

PRIORITY® MAIL
UNITED STATES POSTAL SERVICE
Visit us at usps.com

Label 107R, January 2008