UNITED STATES PATENT AND TRADEMARK OFFICE

________________________
BEFORE THE PATENT TRIAL AND APPEAL BOARD

________________________
INTEL CORPORATION
Petitioner

v.

ALACRITECH, INC.
Patent Owner

________________________
Case IPR. No. Unassigned
U.S. Patent No. 7,237,036
Title: FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING
A TCP CONNECTION

________________________
Petition For Inter Partes Review of U.S. Patent No. 7,237,036 Under
35 U.S.C. §§ 311-319 and 37 C.F.R. §§ 42.1-.80, 42.100-.123

Mail Stop “PATENT BOARD”
Patent Trial and Appeal Board
U.S. Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450
TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>2. REQUIREMENTS FOR PETITION FOR INTER PARTES REVIEW</td>
<td>1</td>
</tr>
<tr>
<td>2.1. Grounds for Standing (37 C.F.R. § 42.104(a))</td>
<td>1</td>
</tr>
<tr>
<td>2.2. Notice of Lead and Backup Counsel and Service Information</td>
<td>1</td>
</tr>
<tr>
<td>2.3. Notice of Real-Parties-in-Interest (37 C.F.R. § 42.8(b)(1))</td>
<td>2</td>
</tr>
<tr>
<td>2.4. Notice of Related Matters (37 C.F.R. § 42.8(b)(2))</td>
<td>3</td>
</tr>
<tr>
<td>2.5. Fee for Inter Partes Review</td>
<td>14</td>
</tr>
<tr>
<td>2.6. Proof of Service</td>
<td>14</td>
</tr>
<tr>
<td>3. IDENTIFICATION OF CLAIMS BEING CHALLENGED (§42.104(B))</td>
<td>14</td>
</tr>
<tr>
<td>4. BACKGROUND OF THE TECHNOLOGY</td>
<td>15</td>
</tr>
<tr>
<td>4.1. TCP/IP</td>
<td>15</td>
</tr>
<tr>
<td>4.2. UDP/IP</td>
<td>21</td>
</tr>
<tr>
<td>4.3. Protocol Offload</td>
<td>22</td>
</tr>
<tr>
<td>5. OVERVIEW OF THE 036 PATENT</td>
<td>23</td>
</tr>
<tr>
<td>6. 036 PATENT PROSECUTION HISTORY</td>
<td>27</td>
</tr>
<tr>
<td>7. CLAIM CONSTRUCTION</td>
<td>31</td>
</tr>
<tr>
<td>7.1. Applicable Law</td>
<td>31</td>
</tr>
<tr>
<td>7.2. Construction of Claim Terms</td>
<td>32</td>
</tr>
<tr>
<td>7.2.1. “context for communication”</td>
<td>32</td>
</tr>
<tr>
<td>7.2.2. “prepend”</td>
<td>33</td>
</tr>
<tr>
<td>8. PERSON HAVING ORDINARY SKILL IN THE ART</td>
<td>33</td>
</tr>
</tbody>
</table>
9. DESCRIPTION OF THE PRIOR ART .............................................................34


9.2. U.S. Patent No. 5,768,618 (“Erickson”) .........................................................39

9.3. Motivations To Combine Erickson and Tanenbaum96 ..........................46

10. GROUND #1: ...............................................................................................50

10.1. Claims 1-7 are unpatentable as obvious over Erickson in view of Tanenbaum96 .................................................................50

10.1.1. [1.P.1] A device for use with a first apparatus that is connectable to a second apparatus .................................50

10.1.2. [1.P.2] the first apparatus containing a memory and a first processor ........................................................................52

10.1.3. [1.P.3] [first processor] operating a stack of protocol processing layers that create a context for communication, the context including a media access control (MAC) layer address, an Internet Protocol (IP) address and Transmission Control Protocol (TCP) state information, the device comprising: ........................................................................54

10.1.4. [1.1] a communication processing mechanism connected to the first processor ..............................................63

10.1.5. [1.2] said communication processing mechanism containing a second processor ..............................................65

10.1.6. [1.3] [second processor] running instructions to process a message packet such that the context is employed to transfer data contained in said packet to the first apparatus memory and ........................................66

10.1.7. [1.4] the TCP state information is updated by said second processor.” .................................................................68
10.2. Claim 2 is unpatentable as obvious over Erickson in view of Tanenbaum96

10.2.1. [2] The device of claim 1, wherein said communication processing mechanism includes a receive sequencer with directions to classify said packet, wherein said packet contains control information corresponding to the stack of protocol layers.

10.3. Claim 3 is unpatentable as obvious over Erickson in view of Tanenbaum96

10.3.1. [3] The device of claim 1, wherein said communication processing mechanism includes a receive sequencer with directions to generate a summary of a second message packet received from the network, said second packet containing control information corresponding to the stack of protocol layers, and said instructions including an instruction to compare said summary with said context.

10.4. Claim 4 is unpatentable as obvious over Erickson in view of Tanenbaum96

10.4.1. [4.1] The device of claim 1, wherein said instructions include a first instruction to create a header corresponding to said context and having control information corresponding to several of the protocol processing layers,

10.4.2. [4.2] and said instructions include a second instruction to prepend said header to second data for transmission of a second packet.

10.5. Claim 5 is unpatentable as obvious over Erickson in view of Tanenbaum96
10.5.1. [5.1] The device of claim 1, wherein said communication processing mechanism has a direct memory access unit to send, based upon said context, said data from said communication processing mechanism to the first apparatus memory, ..............................................................81

10.5.2. [5.2] without a header accompanying said data. ..................84

10.6. Claim 6 is unpatentable as obvious over Erickson in view of Tanenbaum96 .................................................................85

10.6.1. [6] The device of claim 1, wherein said context includes a receive window of space in the memory that is available to store application data, and said communication processing mechanism advertises said receive window. ......................................................85

10.7. Claim 7 is unpatentable as obvious over Erickson in view of Tanenbaum96 ............................................................................87

10.7.1. [7] The device of claim 1, wherein said context includes TCP ports of said first and said second apparatuses........................87

11. CONCLUSION .........................................................................................89
## Exhibit List

<table>
<thead>
<tr>
<th>Exhibit #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ex.1002</td>
<td>Excerpts from Prosecution File History of U.S. Patent No. 7,237,036 (“036 File History”)</td>
</tr>
<tr>
<td>Ex.1003</td>
<td>Declaration of Robert Horst</td>
</tr>
<tr>
<td>Ex.1004</td>
<td><em>Curriculum Vitae</em> of Robert Horst</td>
</tr>
<tr>
<td>Ex.1005</td>
<td>U.S. Patent No. 5,768,618 (“Erickson”)</td>
</tr>
<tr>
<td>Ex.1011</td>
<td>Declaration of Rice Mayors regarding Tanenbaum, Andrew S., <em>Computer Network</em></td>
</tr>
<tr>
<td>Ex.1012</td>
<td>U.S. Patent No. 4,831,523</td>
</tr>
<tr>
<td>Exhibit #</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>Ex.1020</td>
<td>U.S. Patent No. 5,619,650 (“Bach”)</td>
</tr>
<tr>
<td>Ex.1021</td>
<td>U.S. Patent No. 5,915,124 (“Morris”)</td>
</tr>
<tr>
<td>Exhibit #</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>Ex.1031</td>
<td>U.S. Provisional Application 60/061,809 (“Alacritech 1997 Provisional Application”)</td>
</tr>
<tr>
<td>Ex.1032</td>
<td>Culler, E.C., et al., “Parallel Computing on the Berkeley NOW”, Computer Science Division, University of California, Berkeley. (“Culler”)</td>
</tr>
<tr>
<td>Exhibit #</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
1. **INTRODUCTION**

Pursuant to 35 U.S.C. §§ 311-319 and 37 C.F.R. §§ 42.1-.80, 42.100-.123, Intel Corporation (“Petitioner” or “Intel Corporation”) hereby petitions the Patent Trial and Appeal Board to institute an *inter partes* review of claims 1-7 of U.S. Patent No. 7,237,036, titled “Fast-Path Apparatus for Receiving Data Corresponding [sic] a TCP connection” (Ex.1001, the “036 Patent”), and cancel those claims as unpatentable.

More specifically, claims 1-7 relate to the well-known prior art technique of fast TCP protocol processing by a network interface card (NIC).

2. **REQUIREMENTS FOR PETITION FOR INTER PARTES REVIEW**

2.1. **Grounds for Standing (37 C.F.R. § 42.104(a))**

Petitioner certifies that the 036 Patent is available for *inter partes* review and that Petitioner is not barred or estopped from requesting *inter partes* review of the challenged claims of the 036 Patent on the grounds identified herein.

2.2. **Notice of Lead and Backup Counsel and Service Information**

Pursuant to 37 C.F.R. §§ 42.8(b)(3), 42.8(b)(4), and 42.10(a), Petitioner provides the following designation of Lead and Back-Up counsel.

<table>
<thead>
<tr>
<th>Lead Counsel</th>
<th>Back-Up Counsel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Garland T. Stephens</td>
<td>Jeremy Jason Lang</td>
</tr>
<tr>
<td>Registration No. 37,242</td>
<td>Registration No. 73,604</td>
</tr>
<tr>
<td>(<a href="mailto:garland.stephens@weil.com">garland.stephens@weil.com</a>)</td>
<td>(<a href="mailto:jason.lang@weil.com">jason.lang@weil.com</a>)</td>
</tr>
<tr>
<td>Postal &amp; Hand-Delivery Address:</td>
<td>Postal &amp; Hand-Delivery Address:</td>
</tr>
<tr>
<td>Weil, Gotshal &amp; Manges LLP</td>
<td>Weil, Gotshal &amp; Manges LLP</td>
</tr>
</tbody>
</table>
Petitioner consents to service by electronic mail at the following address: Intel.Alacritech.IPR@weil.com. Pursuant to 37 C.F.R. § 42.10(b), a Power of Attorney for Petitioner is attached.

2.3. **Notice of Real-Parties-in-Interest (37 C.F.R. § 42.8(b)(1))**

Petitioner, Intel Corporation, is the real-party-in-interest. No other parties exercised or could have exercised control over this petition; no other parties funded

2.4. Notice of Related Matters (37 C.F.R. § 42.8(b)(2))

The following judicial or administrative matters may affect or be affected by a decision in this proceeding: *Alacritech, Inc. v. CenturyLink, Inc.*, 2:16-cv-00693-JRG-RSP (E.D. Tex.); *Alacritech, Inc. v. Wistron Corp.*, 2:16-cv-00692-JRG-RSP (E.D. Tex.); *Alacritech, Inc. v. Dell Inc.*, 2:16-cv-00695-RWS-RSP (E.D. Tex.). In addition to this Petition, Petitioner is filing petitions for U.S. Patent Nos. 8,805,948; 7,337,241; 7,673,072; 7,124,205; 7,945,699; 8,131,880; and 9,055,104.

Alacritech has asserted one or more of the Asserted Patents or patents related to the Asserted Patents in the following actions:

1. U.S. Patent Nos. 6,427,171 and 6,697,868 are asserted in *Alacritech, Inc. v. Microsoft Corp.*, 3:04-cv-03284, (N.D. Cal);

2. U.S. Patent Nos. 7,124,205; 7,237,036; 7,337,241; 7,673,072; 8,131,880; 8,805,948; and 9,055,104 are asserted in *Alacritech, Inc. v. Wistron Corp.*, 2:16-cv-00692-JRG-RSP (E.D. Tex.);

3. U.S. Patent Nos. 7,124,205; 7,237,036; 7,337,241; 7,673,072; 8,131,880; 8,805,948; 9,055,104; and 7,945,699 are asserted in *Alacritech, Inc. v. Dell Inc.*, 2:16-cv-00695-RWS-RSP (E.D. Tex.);
4. U.S. Patent Nos. 7,124,205; 7,237,036; 7,337,241; 7,673,072; 8,131,880; 8,805,948; 9,055,104; and 7,945,699 are asserted in Alacritech, Inc. v. CenturyLink, Inc., 2:16-cv-00693-JRG-RSP (E.D. Tex.).

The patent family to which the 036 Patent belongs contains 19 additional U.S. patents:

5. U.S. Patent Application No. 09/855,979 (filed May 14, 2001, issued Nov. 7, 2006 as 7,133,940 patent);
6. U.S. Patent Application No. 09/802,426 (filed Mar. 9, 2001, issued May 9, 2006 as 7,042,898 patent);
8. U.S. Patent Application No. 09/802,551 (filed Mar. 9, 2001, issued Jul. 11, 2006 as 7,076,568 patent);


15. U.S. Patent Application No. 09/439,603 (filed Nov. 12, 1999, issued Jun. 12, 2001 as 6,247,060 patent);


19. U.S. Patent Application No. 60/098,296 (expired);

The patent family to which the 205 Patent belongs contains 5 additional U.S. patents:


The patent family to which the 072 Patent belongs contains 19 additional U.S. patents:
5. U.S. Patent Application No. 09/855,979 (filed May 14, 2001, issued Nov. 7, 2006 as 7,133,940 patent);
6. U.S. Patent Application No. 09/802,426 (filed Mar. 9, 2001, issued May 9, 2006 as 7,042,898 patent);
8. U.S. Patent Application No. 09/802,551 (filed Mar. 9, 2001, issued Jul. 11, 2006 as 7,076,568 patent);
15. U.S. Patent Application No. 09/439,603 (filed Nov. 12, 1999, issued Jun. 12, 2001 as 6,247,060 patent);
19. U.S. Patent Application No. 60/098,296 (expired);
The patent family to which the 948 Patent belongs contains 2 additional U.S. patents:

2. U.S. Patent Application No. 09/067,544 (filed Apr. 27, 1998, issued May 1, 2001 as 6,226,680 patent);

The patent family to which the 699 Patent belongs contains 14 additional U.S. patents:

5. U.S. Patent Application No. 09/675,484 (filed Sept. 29, 2000, issued Oct. 19, 2004 as 6,807,581 patent);
13. U.S. Patent Application No. 60/098,296 (expired)
15. U.S. Patent Application No. 60/061,809 (expired);
The patent family to which the 880 Patent belongs contains 6 additional U.S. patents:

6. U.S. Patent Application No. 60/098,296 (expired);
7. U.S. Patent Application No. 09/067,544 (filed Apr. 27, 1998, issued May 1, 2001 as 6,226,680 patent);

The patent family to which the 104 Patent belongs contains 1 additional U.S. patents:


The patent family to which the 241 Patent belongs contains 19 additional U.S. patents:

5. U.S. Patent Application No. 09/067,544 (filed Apr. 27, 1998, issued May 1, 2001 as 6,226,680 patent);
6. U.S. Patent Application No. 60/061,809 (expired);
7. U.S. Patent Application No. 09/384,792 (filed Aug. 27, 1999, issued Aug. 13, 2002 as 6,434,620 patent);
9. U.S. Patent Application No. 60/098,296 (expired);


17. U.S. Patent Application No. 09/802,426 (filed Mar. 9, 2001, issued May 9, 2006 as 7,042,898 patent);


2.5. Fee for Inter Partes Review

The Director is authorized to charge the fee specified by 37 C.F.R. § 42.15(a), and any other required fees, to Deposit Account No. 506499.

2.6. Proof of Service

Proof of service of this Petition on the Patent Owner at the correspondence address of record for the 036 Patent is attached.

3. Identification of Claims Being Challenged (§42.104(B))

Ground #1: Claims 1-7 of the 036 Patent are invalid under (pre-AIA) 35 U.S.C. § 103(a) on the ground that they are obvious over:

a) U.S. Pat. No. 5,768,618, to Erickson, entitled “Method for Performing Sequence of Actions in Device Connected to Computer in Response to Specified Values Being Written Into Snooped Sub Portions of Address Space,” filed on Dec. 21, 1995 and issued on Jun. 16, 1998 (Ex.1005, “Erickson”) in view of

b) Computer Networks, A. Tanenbaum, 3rd ed. (1996) (Ex.1006,
“Tanenbaum96”).

4. **BACKGROUND OF THE TECHNOLOGY**

   This section provides a brief background on the technology at issue, focusing on the TCP, UDP and IP protocols and offloading protocol processing from a host computer to a network interface card (NIC). Petitioner’s declarant Dr. Horst provides a tutorial on the technology and discusses the state of the art as well. Ex.1003, Horst Decl. at Section V.

4.1. **TCP/IP**

   The 036 Patent relates to network interface device that provides a “fast path” that offloads protocol processing of TCP/IP packets from the host to a network interface device. Ex.1001, 036 Patent at Abstract.

   TCP/IP stands for Transmission Control Protocol/Internet Protocol. TCP/IP is the main protocol used for Internet communications – Web pages are served using TCP/IP. By October 14, 1997, the date to which the 036 Patent claims priority, TCP/IP was one of the most popular wide area networking protocols. TCP/IP was standardized in a series of publicly available Request for Comments (RFCs) published by the Internet Engineering Task Force, including RFC 793, entitled “Transmission Control Protocol.” Ex.1007, RFC 793. Free implementations of the protocol were widely available and widely used. Ex.1003, Horst Decl. ¶¶24-26.
TCP/IP consists of two parts: (1) Transmission Control Protocol (TCP), which provides virtual bi-directional connections that provided guaranteed in-order, error-free delivery of arbitrary amounts of data between programs running on different computers over the internet; and (2) Internet Protocol (IP), which provides delivery of datagrams (packets) to any routable Internet address, without any reliability or ordering guarantees. TCP/IP can be transmitted over a variety of physical media, the most common of which is Ethernet. *Id.* ¶¶24-31.

TCP runs on “top” of IP by first dividing application data to be transmitted into segments that become the data payloads of TCP packets and concatenating each payload with a TCP header to form a TCP packet, a process called TCP segmentation. TCP/IP then places the resulting TCP packet (TCP header + payload) into the data payload of an IP packet by concatenating the TCP packet (IP data payload) with an IP header. The TCP packet is thus “encapsulated” in an IP packet. For transmission on an Ethernet network, the IP packet is in turn encapsulated in an Ethernet frame by concatenating a Medium Access Control (MAC) header for Ethernet with the IP packet, which becomes the payload of the Ethernet frame, and concatenating a checksum trailer at the end of the frame. *Id.* ¶¶24-31.

A complete TCP/IP packet for Ethernet will thus consist of a MAC header for an Ethernet frame, which has as its payload an IP packet comprising an IP
header which in turn has as its payload a TCP packet. The TCP packet has its own TCP header followed by TCP payload containing application data to be transmitted. An Ethernet checksum trailer appears at the end. The complete packet thus has an Ethernet header, followed by an IP header, followed by a TCP header, followed by a payload, followed by an Ethernet checksum trailer. *Id.* ¶¶24-31.

As described above, and shown in the figure below, in typical TCP/IP processing, a packet is built from the top down, i.e., each layer encapsulates what it receives from the layer above by concatenating a header. When receiving a packet from the network the layers work in reverse, with each layer stripping its header and providing the resulting packet to the layer above. Separate entities typically perform each layer of the protocol processing. The collection of entities for processing the various layers is often called a “protocol stack.” *Id.* ¶¶27-31.
Starting from the lowest layer, the MAC layer (e.g., Ethernet) handles the actual transmission on a physical medium (e.g. Ethernet cables). The header of this layer, e.g., an Ethernet header, includes a MAC address that is the address of a network interface (e.g., on a computer or router) on a local area network. Ex.1003, Horst Decl. ¶¶28-29.

Above the MAC layer is the Internet layer (IP layer). The IP header includes source and destination IP addresses for identifying a computer at each end of the connection. These addresses are typically routable over the global Internet. For example, one IP address for Google.com is 216.58.216.46. Id. ¶30.

Above the IP layer is the TCP layer. The TCP header includes “port numbers,” corresponding to the end points (e.g. client or server programs) sending
and receiving data on each end of the connection. For example, the usual port number for World Wide Web (HTTP) servers is port 80. *Id.* ¶31.

In sum, sending data over a TCP/IP connection using Ethernet has always required: (1) TCP source and destination port numbers, (2) source and destination IP addresses, and (3) source and destination MAC addresses. *Id.*

To provide guaranteed reliable, in order delivery of arbitrary amounts of data, the TCP layer tracks and acknowledges the sequence of packets, so that the sending TCP layer can re-send lost (and therefore unacknowledged) data without the application having to manage this process. The TCP layer assembles the data from packet payloads in the proper order by using sequence numbers. *Id.* ¶¶31, 39-41. The TCP layer also manages a “sliding window” each-end of the connection advertises a “window” to the other indicating how many bytes of data it is prepared to accept so that senders do not overwhelm receivers with too much data, causing packets to be lost. *Id.* ¶¶46-47.

Considerable information must be tracked to maintain an open TCP connection, including sequence numbers, acknowledgement numbers, the sliding windows, TCP, IP and MAC addresses and more. Typically, the state information needed to maintain a TCP connection is held in a connection record also called the Transmission Control Block (TCB) in RFC 793, and TCB is maintained for each connection. Ex.1007, RFC 793 at .024; Ex.1003, Horst Decl. ¶¶31, 39-41, 46-47.
The combination of an IP address and TCP port number is often called a “socket.” A TCP connection can be formed with a pair of sockets, which includes a source IP address and TCP port number and a destination IP address and TCP port number. Establishing a connection over TCP is sometimes called “opening a socket.” *Id.* ¶¶32-35. To open a socket, an IP address and TCP port for an application endpoint at the other end of the connection are specified by the application opening the socket. For instance, a browser accessing Google.com could open a socket to IP address 216.58.216.46 and TCP port 80. The host protocol stack then identifies a MAC address on the local area network for a “next hop” on the route to the address for 216.58.216.46, such as a gateway router that connects the local area network to the Internet. The host exchanges various control packets (using the IP addresses, TCP port and state information, and MAC addresses) with the protocol stack on the destination computer at 216.58.216.46 to open a TCP connection with web server software on the computer that responds to port 80, and enters an ESTABLISHED state. The applications can then exchange data through the TCP connection. *Id.* ¶¶33-35.

Each user application typically has one or more areas of host memory in which it can (1) place data for transmission so that the protocol stack can retrieve it, encapsulate it in packets, and transmit it, and (2) receive data from the network
placed there by the protocol stack (after stripping the MAC, IP and TCP headers from the packet). *Id.* ¶¶36-38, 90-92.

4.2. **UDP/IP**

Internet Protocol supports a second transport-layer protocol, the User Datagram Protocol (UDP). UDP is a connectionless protocol with no reliability or ordering guarantees. *Ex.1006, Tandenbaum96* at .539. Because UDP is connectionless, no setup or teardown packets are required before sending data. *Ex.1003, Horst Decl.* ¶110. It is often used by client-server applications that use one request and one response instead of going through the trouble of establishing and later releasing a connection. Like TCP, UDP uses source and destination port numbers to identify the end points on the machines identified by the IP addresses in a UDP/IP packet. *Ex.1006, Tanenbaum96* at .560.

The relationship between UDP and TCP in the TCP/IP protocol stack is illustrated in the following figure from *Ex.1006, Tanenbaum96, Fig. 1-19* at .055:

![Diagram of TCP/IP protocol stack](image)

*Fig. 1-19. Protocols and networks in the TCP/IP model initially.*

*See also Ex.1003, Horst Decl.* ¶110.
4.3. Protocol Offload

Protocol processing on a host computer requires that the host perform several operations on the data. To increase performance and reduce the demands on a host computer, many prior art solutions offload some (partial offload), or all (full offload), of this processing to a separate device, e.g., a network interface controller (NIC). Ex.1003, Horst Decl. at Section V.C.-E.

As early as 1974, front-end protocol offload was already being considered for standardization as described in request-for-comments RFC 647, which describes a broad consensus that front-ending (i.e. protocol offloading) was desirable. Id. at Section V.C. RFC 929, published in 1984, describes several motivations for offloading protocol processing to an outboard processor:

There are two fundamental motivations for doing outboard processing. One is to conserve the Hosts’ resources (CPU cycles and memory) in a resource sharing intercomputer network, by offloading as much of the required networking software from the Hosts to Outboard Processing Environments (or "Network Front-Ends") as possible. The other is to facilitate procurement of implementations of the various intercomputer networking protocols for the several types of Host in play in a typical heterogeneous intercomputer network, by employing common implementations in the OPE. A third motivation, of basing a network security approach on trusted mandatory OPEs, will not be dealt with here, but is at least worthy of mention.

Ex.1009, RFC 929 at .002; Ex.1003, Horst Decl. at Section V.C.2. RFC 929
identifies many protocols for offloading, including TCP and UDP, and contemplates a wide range of protocol processing to be offloaded, specified by a “mediation level parameter,” ranging from 9 for full offload, 8-1 for various levels of partial offloads, to 0 for no offload. Ex.1009, RFC 929 at .015-.016; Ex.1003, Horst Decl. at Section V.C.

Between the publication of RFC 929 in 1984 and the 1997 provisional application to which the 036 Patent claims priority, a great deal of work was published in the area of protocol offloading. See Ex.1003, Horst Decl. Section V.D-G. This work specifically teaches the alleged inventions claimed in the 036 Patent.

5. OVERVIEW OF THE 036 PATENT

The 036 Patent relates to offloading TCP protocol processing from a host onto a network interface card (NIC). Ex.1001, 036 Patent at Abstract. The specification of the 036 Patent refers to the disclosed NIC as an “intelligent network interface card (INIC)”. See id. The INIC permits two modes of

---

1 Petitioner does not concede that the 036 Patent claims are entitled the October 14, 1997 priority date.

operation: a “fast path” in which protocol processing from the physical layer through the TCP layer is performed on the INIC, and a “slow path” in which network frames are handed to the host at the MAC layer and passed up through the host protocol stack conventionally. The concept is illustrated in Fig. 24, shown below:

![Diagram](image)

**FIG. 24**

The answer shown in FIG. 24 is to use two modes of operation: One in which the network frames are processed on the INIC through TCP and one in which the card operates like a typical dumb NIC. We call these two modes fast-path, and slow-path. In the slow-path case, network frames are handed to the system at the MAC layer and passed up through the host protocol stack like any other network frame. In the fast path case, network data is given to the host after the headers have been processed and stripped.
The transmit case works in much the same fashion. In slow-path mode the packets are given to the INIC with all of the headers attached. The INIC simply sends these packets out as if it were a dumb NIC. In fast-path mode, the host gives raw data to the INIC which it must carve into MSS sized segments, add headers to the data, perform checksums on the segment, and then send it out on the wire.

Ex.1001, 036 Patent at 39:10-27, Fig. 24.

The INIC uses a “connection context” to determine which “path” should be used for a received packet:

The IP source address of the IP header, the IP destination address of the IP header, the TCP source address of the TCP header, and the TCP destination address of the TCP header together uniquely define a single connection context (TCB) with which the packet is associated. Processor 470 examines these addresses of the TCP and IP headers and determines the connection context of the packet. Processor 470 then checks a list of connection contexts that are under the control of INIC card 200 and determines whether the packet is associated with a connection context (TCB) under the control of INIC card 200.

If the connection context is not in the list, then the "fast-path candidate" packet is determined not to be a "fast-path packet." In such a case, the entire packet (headers 20 and data) is transferred to a buffer in host 20 for "slow-path" processing by the protocol stack of host 20.
The “context” for each connection “summarize[es] various features of the connection.” *Id.* at 7:62-66, 8:3-15, 10:19-22. The host may create the context by processing an initial request packet, e.g., as part of opening a connection. *Id.* at 10:19-22.

Claim 1 of the 036 Patent recites:

1. A device for use with a first apparatus that is connectable to a second apparatus, the first apparatus containing a memory and a first processor operating a stack of protocol processing layers that create a context for communication, the context including a media access control (MAC) layer address, an Internet Protocol (IP) address and Transmission Control Protocol (TCP) state information, the device comprising:

   a communication processing mechanism connected to the first processor, said communication processing mechanism containing a second processor running instructions to process a message packet such that the context is employed to transfer data contained in said packet to the first apparatus memory and the TCP state information is updated by said second processor.

*Id.* at Claim 1.

---

3 Emphasis added unless otherwise noted.
Independent claim 1 is “a device for use with a first apparatus that is connectable to a second apparatus.” The second apparatus appears nowhere else in the claim. The first apparatus contains a memory and a first processor operating a protocol stack. The protocol stack creates a “context for communication” including a MAC address, IP address and TCP state information.

The device of claim 1 comprises a communication processing mechanism containing a second processor that processes packets, and employs a “context” to transfer data to the first apparatus memory. The second processor updates TCP state information in the context.

Claims 2-7 depend from claim 1. Claims 2-3 relate to classifying packets for either the slow and fast path. Claim 4 relates to the NIC creating headers in connection with sending data on the fast-path. Claim 5 adds using direct memory access (DMA) to write the data from the NIC to the host. Claims 6-7 relate to the context including a TCP receive window and TCP ports.

6. **036 PATENT PROSECUTION HISTORY**

This summary focuses on the claims 1-7 (the challenged claims). In this summary, claims 1-7 correspond to issued claims 1-7.

On September 27, 2002, the application that matured into the 036 Patent was filed. On November 16, 2005, the Examiner rejected claims 1-7 under 35 U.S.C. §
102(a) as being anticipated by U.S. Pat. No. 6,122,670 ("Bennett"). Ex.1002, 036 File History at .259-.263.

On March 31, 2006, Applicant amended claims 1 and 6-7 as follows:

1. (currently amended) A device for use with a first apparatus that is connectable to a second apparatus by a network, the first apparatus containing a memory and a first processor operating a stack of protocol processing layers that create a context for communication between an application of the first apparatus and an application of the second apparatus, the context including a media access control (MAC) layer address, an Internet Protocol (IP) address and Transmission Control Protocol (TCP) state information, the device comprising:

   a communication processing mechanism connected to the first processor and to the network, said communication processing mechanism containing a second processor and running instructions to choose, by referencing the context, whether a network message packet is processed by the protocol processing layers or the context is employed to transfer data contained in said packet between the network and the first apparatus memory process a message packet such that the context is employed to transfer data contained in said packet to the first apparatus memory and the state information is updated by said second processor.

6. (currently amended) The device of claim 1, wherein said instructions include receive instructions to process packets received by the first apparatus from the network and transmit instructions to process packets transmitted from the first apparatus to the network context includes a receive window of space in the memory that is available to store application data, and said communication processing mechanism advertises said receive window.

7. (currently amended) The device of claim 1, wherein said context includes Internet Protocol (IP) addresses of said first and said second apparatuses and Transport Control Protocol (TCP) TCP ports of said first and said second apparatuses.

_Id._ at .273-.274.
Minor amendments were also made to claims 3-4. *Id.* Applicant then argued that Bennett was not enabling. *Id.* at .280-.283. Finally, Applicant attempted to distinguish Bennett as follows:

> Applicants respectfully assert that Bennett does not disclose a “communication processing mechanism containing a second processor running instructions to process a message packet such that the context is employed to transfer data contained in said packet to the first apparatus memory and the state information is updated by said second processor,” in contrast to claim 1.

> Applicants respectfully assert that Bennett does not disclose that the context includes a media access control (MAC) layer address, in contrast to claim 1.

> For at least these reasons, claim 1 and all the claims that depend from claim 1 are not anticipated by Bennett.

*Id.* at .284.

On July 5, 2006, the Examiner rejected claim 1 under 35 U.S.C. § 103(a) as being obvious over Bennett in view of U.S. Pat. No. 6,195,739 (“Wright”) and objected to claims 2-7 as being dependent upon a rejected base claim. *See id.* at .292-.294, .295. The Examiner stated that while Bennett did not disclose “running instructions to process a message packet such that the context is employed to transfer data contained in said packet to the first apparatus memory and the state information is updated by said second processor,” Wright did. *See id.* at .292-.293.

On October 10, 2006, Applicant amended claim 1 as follows:
Applicant argued that Bennett was not enabled, id. at .309-.310, and that Wright was filed after the effective filing date of the 036 Patent (alleging that the effective filing date of the 036 Patent is October 14, 1997), id. at .310. Next, Applicant argued that Wright’s disclosures relate to operation underneath the TCP layer, and thus it does not disclose “the TCP state information is updated” or “that the context is employed to transfer data contained in said packet to the first apparatus memory.” Id. at .311. Finally, Applicant argued that one of ordinary skill in the art would not have combined Bennett and Wright, and that even if they were combined, there would still be nonobvious differences over the combination. Id.
On February 7, 2007, the Examiner issued a notice of allowance, allowing, among others, claims 1-7. Id. at .325. The notice did not describe the reason for the allowance. Id. at .326-.327.

7. CLAIM CONSTRUCTION

7.1. Applicable Law

The “broadest reasonable construction in light of the specification of the patent in which it appears” applies to the 036 Patent. 37 C.F.R. § 42.100(b). Any ambiguity regarding the “broadest reasonable construction” of a claim term is

---

4 Petitioner expressly reserves the right to challenge in district court litigation one or more claims (and claim terms) of the 036 Patent for failure to satisfy the requirements of 35 U.S.C. § 112, which cannot be raised in these proceedings. See 35 U.S.C. § 311(b). Nothing in this Petition shall be construed as a waiver of such challenge, or agreement that the requirements of 35 U.S.C. § 112 are met for any claim of the 036 Patent.

5 The district court, in contrast, applies the Phillips standard. Petitioner expressly reserves the right to argue different or additional claim construction positions under this standard in district court. Note that the 036 patent will not expire during the pendency of this IPR due to a term extension.

7.2. Construction of Claim Terms

All claim terms not specifically addressed in this Section have been accorded their broadest reasonable interpretation as understood by a person of ordinary skill in the art and consistent with the specification of the 036 Patent. Petitioner respectfully submits that the following terms shall be construed for this IPR:

7.2.1. “context for communication”

Under the Phillips claim construction standard, Patent Owner-Alacritech has taken the position in the copending district court litigation that “context” means “data regarding an active connection.” Ex.1010, Patent Owner Alacritech’s Preliminary Claim Construction Disclosures, at .001; see also Ex.1039, JCCS at .059-.063 (alleged intrinsic support for the construction).

Also under the Phillips claim construction standard in the co-pending litigation, Petitioner contends that “context” is indefinite as used in the 036 Patent claims because the specification and file history fail to inform, with reasonable certainty, the proper scope of “context,” and in particular, its bounds. See also Ex.1039 at .055.
Under the broadest reasonable construction standard, this Petition applies Patent Owner’s district construction, which is “data regarding an active connection.”

7.2.2. **“prepend”**

The term “prepend” appears in claim 4 of the 036 Patent. Under the broadest reasonable construction standard, this term in light of the specification would have been understood to mean “add to the front.” See Ex.1003, Horst Decl. ¶108. The specification specifically defines “prepends” in this manner: “Once the packet control sequencer 176 detects that all of the packet has been processed by the fly-by sequencer 178, the packet control sequencer 176 … prepends (adds to the front) that status information to the packet …” Ex.1001, 036 Patent at 14:5-12.

In the copending litigation, the Patent Owner contends that no construction is necessary for “prepend.” Ex.1010, Patent Owner Alacritech’s Preliminary Claim Construction Disclosures at .005.

8. **PERSON HAVING ORDINARY SKILL IN THE ART**

A person having ordinary skill in the art (“POSA”) with respect to the technology described in the 036 Patent would be a person with at least the equivalent of a B.S. degree in computer science, computer engineering or electrical engineering with at least five years of industry experience including experience in
computer architecture, network design, network protocols, software development, and hardware development. See Ex.1003, Horst Decl. ¶¶18-19.

When this Petition refers to the understanding of a POSA, e.g., what a POSA would have understood, what a POSA understands, that a POSA would have been motivated, etc., it is with respect to what the POSA would have understood in October 14, 1997, the date of the provisional to which the 036 Patent claims priority.

9. DESCRIPTION OF THE PRIOR ART


Tanenbaum96, “Computer Networks,” is a widely-cited textbook covering network hardware, software, protocols and standards. It was published in 1996 and

\[^6\] Tanenbaum96 is prior art under at least 35 U.S.C. § 102(e). Ex.1006; see also Ex.1011 (declaration of librarian). Tanenbaum96 was cited by Applicant, along with roughly 100 additional references, during the prosecution of the 036 Patent; it was not discussed in the prosecution. Applicant incorporates and references Tanenbaum96 as background material in describing networks, illustrating that it is prior art and that it was a well-known resource to a POSA. See Ex.1001, 036 Patent at 4:47-63.
is therefore prior art to the October 14, 1997 provisional application to which the 036 Patent claims priority. See also generally Ex.1003, Horst Decl. ¶¶109-122.

Tanenbaum96 recognizes that an “obstacle to fast networking is protocol software,” and teaches “fast path” processing for TCP as a solution. Ex.1006, Tanenbaum96 at .583-.585. Tanenbaum96 teaches fast path transmissions using a prototype header stored in the transport entity, because in the normal case of an established TCP connection, only a few fields of the header change in consecutive packets:

The first thing the transport entity does is make a test to see if this is the normal case: the state is ESTABLISHED, neither side is trying to close the connection, a regular (i.e., not an out-of-band) full TPDU [Transport Protocol Data Unit, i.e. packet] is being sent, and there is enough window space available at the receiver. If all conditions are met, no further tests are needed and the fast path through the sending transport entity can be taken.

In the normal case, the headers of consecutive data TPDUs are almost the same. To take advantage of this fact, a prototype header is stored within the transport entity. At the start of the fast path, it is copied as fast as possible to a scratch buffer, word by word. Those fields that change from TPDU to TPDU are then overwritten in the buffer.

Ex.1006, Tanenbaum96 at .583.

Tanenbaum96 teaches that the transport entity may be built into the network interface card:
The hardware and/or software within the transport layer that does the work is called the transport entity. The transport entity can be in the operating system kernel, in a separate user process, in a library package bound into network applications, or on the network interface card.

Id. at .498 (underlining added, bold in original).

Tanenbaum96 goes on to describe a TCP prototype header in detail:

As an example of how this principle works in practice, let us consider TCP/IP. Fig. 6-50(a) shows the TCP header. The fields that are the same between consecutive TPDUs on a one-way flow are shaded. All the sending transport entity has to do is copy the five words from the prototype header into the output buffer, fill in the next sequence number (by copying it from a word in memory), compute the checksum, and increment the sequence number in memory. It can then hand the header and data to a special IP procedure for sending a regular, maximum TPDU. IP then copies its five-word prototype header [see Fig. 6-50(b)] into the buffer, fills in the Identification field, and computes its checksum. The packet is now ready for transmission.

Ex.1006, Tanenbaum96 at .584 (emphasis added).

Tanenbaum96 also teaches TCP fast path receiving by looking up a TCP connection record based on the IP source address, TCP source port, IP destination address and TCP destination address, checking to see if it the packet is a normal
one in the ESTABLISHED state (and thus suitable for fast-path processing), and then putting the data into user memory:

Now let us look at fast path processing on the receiving side…. For TCP, the connection record can be stored in a hash table for which some simple function of the two IP addresses and two ports is the key. Once the connection record has been located, both addresses and both ports must be compared to verify that the correct record has been found…

The TPDU [Transport Protocol Data Unit, i.e. packet] is then checked to see if it is a normal one: the state is ESTABLISHED, neither side is trying to close the connection, the TPDU is a full one, no special flags are set, and the sequence number is the one expected. These tests take just a handful of instructions. If all conditions are met, a special fast path TCP procedure is called.

The fast path updates the connection record and copies the data to the user. While it is copying, it also computes the checksum, eliminating an extra pass over the data. If the checksum is correct, the connection record is updated and an acknowledgement is sent back. The general scheme of first making a quick check to see if the header is what is expected, and having a special procedure to handle that case, is called header prediction. Many TCP implementations use it.

Id. at .584-.585 (underlining added, bold in original).

The “connection record” disclosed in Tanenbaum96 is used to maintain TCP state:
When an application on the client machine issues a CONNECT request, the local TCP entity creates a connection record, marks it as being in the *SYN SENT* state, and sends a *SYN* segment. Note that many connections may be open (or being opened) at the same time on behalf of multiple applications, so the state is per connection and recorded in the connection record.

*Id.* at .549. This is the same as the “Transmission Control Block (TCB)” described in RFC 793, the TCP protocol specification:

Before we can discuss very much about the operation of the TCP we need to introduce some detailed terminology. The maintenance of a TCP connection requires the remembering of several variables. We conceive of these variables being stored in a connection record called a Transmission Control Block or TCB.

Ex.1007, RFC 793 at .024.

Tanenbaum96 teaches that “[f]or TCP, the connection record can be stored in a hash table for which some simple function of the two IP addresses and two ports is the key.” Ex.1006, Tanenbaum96 at .585. Tanenbaum96 thus teaches the “connection context” as described in the 036 Patent:

IP source address of the IP header, the IP destination address of the IP header, the TCP source address of the TCP header, and the TCP destination address of the TCP header [that] together uniquely define a single connection context (TCB) with which the packet is associated.
Ex.1001, 036 Patent 31:7-12. Like the context of claim 1 of the 036 Patent, the connection record of Tanenbaum96 is used to transfer data to host memory. Ex.1006, Tanenbaum96 at .584-.585.

Tanenbaum96 does not explicitly teach that the connection record includes a MAC address. Erickson, discussed next, does.

9.2. U.S. Patent No. 5,768,618 (“Erickson”) 7

Erickson describes a fast path for network connections in which an I/O adapter performs fast path network protocol processing, including for UDP/IP and TCP/IP. See also generally Ex.1003, Horst Decl. ¶¶123-137.

Specifically, Erickson discloses an I/O Device Adapter 314 to offload protocol processing for fast applications as shown in Fig. 3:

7 Erickson was filed on Dec. 21, 1995 and issued on Jun. 16, 1998, and is therefore prior art at least under § 102(e). See Ex.1005, Erickson. Erickson was not cited to or by the Examiner during prosecution of the 036 Patent.
FIG. 3 is a flow diagram describing the system data flow of fast and slow applications 302, 304, and 306 compatible with the present invention. A traditional slow application 306 uses normal streams processing 308 to send information to a pass-through driver 310. The pass-through driver 310 initializes the physical hardware registers 320 of the I/O device adapter 314 to subsequently transfer the information through the I/O device adapter 314 to the commodity interface 322. With the present invention, fast user applications 302 and 304 directly use a setup driver 312 to initialize the physical hardware registers 320, then send the information directly through the I/O device adapter 314 to the commodity interface 322 via virtual hardware 316 and 318. Thus, the overhead of the normal streams processing 308 and pass-through driver 310 are eliminated with the use of the virtual hardware 316 and 318 of the present invention, and fast applications 302 and...
304 are able to send and receive information more quickly than slow application 306.

*Id.* at 4:53-5:3.

User processes that wish to communicate over the network open a device driver, and specify the details of the desired communication mode. The device driver sets up a protocol script and protocol specific endpoint data for the connection:

Typically, when a user process opens a device driver, the process specifies its type, which may include, but is not limited to, a UDP datagram, source port number, or register address. The user process also specifies either a synchronous or asynchronous connection. The device driver sets up the registers 508 and 504, endpoint table 514, and endpoint protocol data 518. The protocol script 516 is typically based upon the endpoint data type, and the endpoint protocol data 518 depends on protocol specific data.

*Id.* at 6:1-9.

The endpoint data is stored on the I/O device adapter, which uses it to move data from the adapter to user memory:

The I/O device adapter implementation includes a software register 508 and a physical address buffer map 510 in the adapter's memory 512. An endpoint table 514 in the memory 512 is used to organize multiple memory pages for individual user processes. Each entry within the endpoint table 514 points to various protocol data 518 in the memory 512 in order to accommodate multiple communication
protocols, as well as previously defined protocol scripts 516 in the memory 512, which indicate how data or information is to be transferred from the memory 512 of the I/O device adapter to the portions of main memory 502 associated with a user process.

Id. at 5:56-67.

User processes also pre-negotiate packet protocol and state information including MAC, IP and transport layer (UDP) headers. Ex.1005, Erickson at 6:57-7:4, see also Figs. 6 and 7. Specifically, Erickson discloses an exemplary pre-negotiation of transport-layer UDP/IP/MAC protocol information:

Each user process has basically pre-negotiated almost everything about the datagram 602, except the actual user data 610. This means most of the fields in the three header areas 604, 606, and 608 are predetermined.

In this example, the user process and the device driver has pre-negotiated the following fields from FIG. 6: (1) Ethernet Header 604 (Target Ethernet Address, Source Ethernet Address, and Protocol Type); (2) IP Header 606 (Version, IP header Length, Service Type, Flag, Fragment Offset, Time_to_Live, IP Protocol, IP Address of Source, and IP Address of Destination); and (3) UDP Header 608 (Source Port and Destination Port). Only the shaded fields in FIG. 6, and the user data 610, need to be changed on a per-datagram basis.

Id. at 6:57-7:4, see also Figs. 6 and 7.
Erickson discloses that after the pre-negotiation, the I/O device adapter runs protocol scripts to process outgoing and incoming data packets, thereby offloading the protocol processing onto the I/O device adapter. *Id.* at 4:18-23. Erickson describes scripts for a variety of protocols including TCP/IP. The scripts are used to locate an application endpoint and to generate packet headers from a pre-negotiated template header.

Protocol scripts typically serve two functions. The first function is to describe the protocol the software application is using. This includes but is not limited to how to locate an application endpoint, and how to fill in a protocol header template from the application specific data buffer. The second function is to define a particular set of instructions to be performed based upon the protocol type. Each type of protocol will have its own script. Types of protocols include, but are not limited to, TCP/IP, UDP/IP, BYNET lightweight datagrams, deliberate shared memory, active message handler, SCSI, and File Channel.

*Id.* at 5:41-51.

Erickson provides a detailed example for UDP (a transport protocol that, like TCP/IP, runs on top of IP) and Erickson directs the user to the 1981 edition of Tanenbaum’s Computer Networks textbook for details on TCP. *Id.* at 4:38-43. The 1996 edition of Tanenbaum is discussed above, and was the current version of
the textbook as of October 14, 1997, the date of the provisional application to which the 036 Patent claims priority.

Erickson discloses fast-path receiving of data packets by directly writing the data to the host memory space corresponding to the user process (i.e., the fast application), bypassing the protocol stack on the host. *Id.* at 5:53-67. The transfer of data directly to the host memory (and from the host memory to the I/O device adapter) occurs via a Direct Application Interface (DAI):

![FIG. 4](image)

FIG. 4 is a block diagram describing a direct application interface (DAI) and routing of data between processes and an external data connection which is compatible with the present invention. Processes 402 and 404 transmit and receive information directly to and from an interconnect 410 (e.g., I/O device adapter) through the DAI interface.
408. The information coming from the interconnect 410 is routed
directly to a process 402 or 404 by use of virtual hardware and
registers, rather than using a traditional operating system interface
406.

_id._ at 5:5-5:14.

Erickson also discloses fast-path transmission, in which the user process
identifies a block of raw data to be transmitted and “spans” (i.e. sets to 1) a GO
register to trigger the adapter to take the raw data, encapsulate it into a packet with
UDP, IP and MAC headers, and transmit it. See id. at 7:39-47.

Erickson’s network interface device creates headers for packets to be
transmitted using the pre-negotiated UDP, IP and MAC header information. A
user program (senduserinfo at 7:22) identifies raw data in host memory to
be transmitted (by providing a USERDATA_ADDRESS and
USERDATA_LENGTH) and then triggers the network interface device (by
“spanking” the GO register) as shown at Ex.1005, Erickson at 7:18-35. In
response, the network interface device executes a UDP protocol script (udpscript
at 7:51) that creates headers from the pre-negotiated context by populating
UDP/IP/MAC datagram template headers as shown in Fig. 7 with appropriate
values for IP Length, IP Datagram ID, IP Checksum, UDP Length and UDP
Checksum. The network interface device then encapsulates the data with the
headers, and sends the completed packet. Id. at 7:39-64.
9.3. Motivations To Combine Erickson and Tanenbaum96

Erickson provides an express motivation to the user to consult the 1981 Tanenbaum textbook for details on TCP, and incorporates it by reference. *Id.* at 4:34-43. Tanenbaum96 is the third edition of the 1981 Tanenbaum book. Ex.1006, Tanenbaum96 at .004. The 1996 version was the current edition as of the October 14, 1996 critical date for the October 14, 1997 provisional application from which the 036 Patent claims priority. A POSA working on October 13, 1996 would have been motivated to consult Tanenbaum96, the then-current edition of the Tanenbaum textbook cited by Erickson. Ex.1003, Horst Decl. ¶¶138-39.

In 1996, with the growth of the Internet and World Wide Web, TCP/IP was becoming very popular. Ex.1003, Horst Decl. ¶¶26, 140. Accordingly, a POSA would have been motivated to implement the TCP/IP fast path protocol processing described by Erickson, using Erickson’s Ethernet I/O device adapter, and would have further been motivated to consult Tanenbaum96 to do so. *Id.* ¶140. In fact, Erickson expressly references TCP/IP scripts, providing another express motivation to consult Tannenbaum96 to implement a TCP/IP connection on the Erickson Ethernet I/O device adapter. Ex.1005, Erickson at 5:41-51. Indeed, given TCP/IP’s popularity among the finite number of networking protocols, it would have been obvious to try, i.e., look to Tanenbaum96 and accordingly try to implement a TCP/IP connection on the Erickson I/O device adapter. *Id.*, ¶140.
Further, given that a POSA would have understood TCP/IP well, and that the standards for TCP/IP are set forth in well-known Request for Comments (RFCs), a POSA would have had a high expectation of success in implementing TCP/IP on Erickson’s I/O device adapter. *Id.* ¶¶140-41.

A POSA would have readily understood that Tanenbaum96’s teachings of fast path TCP processing using a prototype header, connection records, and header prediction correspond to, and could be used to modify Erickson’s endpoint information, pre-negotiated protocol information, template header and UDP script to perform TCP protocol processing. *Id.* ¶¶142-44.

For transmission, Erickson teaches an adapter script that performs fast path UDP/IP processing by updating UDP/IP protocol information including IP Length, IP Datagram ID, IP Checksum, UDP Length and UDP Checksum. Ex.1005, Erickson at 7:48-63. Tanenbaum96 also teaches an adapter that performs fast path TCP/IP protocol processing by updating the same IP protocol information, including IP Length, IP Datagram ID, IP Checksum, and directly corresponding TCP information including the TCP Sequence number and TCP Checksum. Ex.1006, Tanenbaum96 at .584; Ex.1003, Horst Decl. ¶¶141-43. A POSA would therefore be motivated to use Tanenbaum96’s teachings of a TCP prototype header to implement Erickson’s fast path TCP transmit processing. *Id.*
For receiving, Erickson discloses an endpoint table and protocol scripts which store protocol state information and indicate how data is to be transferred from the network interface device to portions of main memory associated with a user process. Ex.1005, Erickson at 5:59-67. A POSA would understand that Erickson’s endpoint table, pre-negotiated protocol information and protocol scripts correspond to Tanenbaum96’s connection records, and that Erickson’s looking up endpoint protocol information in the endpoint table corresponds to Tanenbaum96’s looking up a connection record to copy data to the user after a quick check that the packet is what is expected (header prediction). Ex.1006, Tanenbaum96 at .584-.585. A POSA would therefore be motivated to use the Tanenbaum96 teachings of header prediction to provide Erickson’s fast path receive processing for TCP. Ex.1003, Horst Decl. ¶146.

Combining Tanenbaum96’s TCP/IP and header prediction with Erickson would have been understood as combining known methods to yield predictable results. As Dr. Horst explains, the TCP/IP protocols and fast path TCP processing that Tanenbaum96 teaches were well known, were directly applicable to Erickson’s I/O device adapter, and a POSA would have had a high expectation of success in adapting the Erickson’s I/O device adapter to offload a TCP/IP connection and further to use Tanenbaum96’s header prediction as the bypass test. Ex.1003, Horst Decl. ¶¶141-44, 147. In other words, the Tanenbaum96 header
prediction was known technique, and would have been understood as a predictable way to improve Erickson’s I/O device adapter.

Moreover, Tanenbaum96 teaches that by using its header prediction teachings, it reduces the complexity of the offloading device (it only has to handle data transfer packets and not set up the connection itself), thereby motivating a POSA to use its header prediction. Ex.1006, Tanenbaum96 at .583 (“The key to fast TPDU processing is to separate out the normal case (one-way data transfer) and handle it specially. Although a sequence of special TPDUs are needed to get into the ESTABLISHED state, once there, TPDU processing is straightforward until one side starts to close the connection.”). In other words, Tanenbaum96’s offloading teachings reduces the complexity of the offloading device, which would therefore reduce the complexity and cost of the Erickson I/O device. Ex.1003, Horst Decl. ¶148.

Further, as part of its header prediction teachings, Tanenbaum96 specifically teaches that connection records (corresponding to Erickson’s endpoint table, pre-negotiated protocol information and protocol scripts) can be stored in a “hash table for which some simple function of the two IP addresses and two ports is the key.” Ex.1006, Tanenbaum96 at .584-.585. That is, Tanenbaum96 details a lookup technique using a “simple function” to implement the bypass test. Ex.1003, Horst Decl. ¶149.
10. **GROUND #1:**

As set forth below, Erickson in view of Tanenbaum96 renders obvious claims 1-7 of the 036 Patent. Erickson discloses the large majority of the limitations through its description of fast path protocol processing of UDP/IP by an I/O device adapter. Tanenbaum96 discloses corresponding details for TCP/IP. In view of Tanenbaum96, it would be obvious to provide the fast path TCP/IP protocol processing disclosed in Erickson, thereby rendering obvious claims 1-7 of the 036 Patent. *See Section 9.3 (motivations to combine).*

10.1. **Claims 1-7 are unpatentable as obvious over Erickson in view of Tanenbaum96**

10.1.1. **[1.P.1] A device for use with a first apparatus that is connectable to a second apparatus**

To the extent that the preamble is limiting, Erickson discloses a *device for use with a first apparatus that is connectable to a second apparatus.*

Specifically, Erickson discloses an “I/O device adapter 314” (*a device*) that is *for use with*, the host “computer” (*a first apparatus*) and a “receiver” computer (*a second apparatus*), wherein the computer and receiver are *connectable* over a network. Ex.1005, Erickson at 1:63-67 ("To overcome the limitations in the prior art described above … the present invention discloses a method of controlling an input/output (I/O) device connected to a computer to facilitate fast I/O data...")
transfers.”), id. at 3:23-36 (“FIG. 1. is a flow diagram illustrating a conventional I/O data flow between a sender and a receiver.”).\footnote{Note that Erickson also refers to the “computer” as a “sender” and “receiver” when discussing the computer sending data (sender) and receiving data (receiver). Id. at 3:23-36}

Erickson discloses that the sender, i.e., the host computer, includes an Ethernet I/O device adapter with a fast path for network protocol processing. See id. at 1:63-67, 4:53-5:5. It connects (is “connectable”) over a conventional network to a second apparatus using conventional network protocols including TCP/IP. Id. at 4:28-43.

These components, as well as components in subsequent claim limitations, are shown below:
Accordingly, Erickson in view of Tanenbaum discloses a device (I/O device adapter 314) for use with a first apparatus (host computer, i.e., sender) that is connectable to a second apparatus (second computer, i.e., receiver).

10.1.2. [1.P.2] the first apparatus containing a memory and a first processor

To the extent that the preamble is limiting, Erickson discloses the first apparatus containing a memory and a first processor.

A POSA would understand that Erickson discloses that the computer (the first apparatus) includes main memory and a processor (containing a memory and a first processor). Ex.1005, Erickson at 1:63-67, see also id. at 9:48 (“memory of computer”) and Fig. 5 (“main memory”); see also Ex.1003, Horst Decl. at
Appendix A, 1.P.2. A computer with main memory, for example, was understood to include a processor. See id.

To the extent that Erickson does not expressly disclose a “processor,” it would have been obvious to a POSA because Erickson discloses that the computer includes applications, meaning that the computer has a processor (and a memory) to execute the applications. See Ex.1005, Erickson at 2:54-61 (operating system and user processes), 6:1-9 (device drivers and user processes); see also Ex.1003, Horst Decl. at Appendix A, 1.P.2.

These components, as well as components in subsequent claim limitations, are shown below:
Ex.1005, Erickson at Fig. 3 (annotated).

Accordingly, Erickson in view of Tanenbaum96 discloses the first apparatus (host computer) containing a memory (e.g., main memory of host computer) and a first processor (its CPU).

10.1.3. [1.P.3] first processor operating a stack of protocol processing layers that create a context for communication, the context including a media access control (MAC) layer address, an Internet Protocol (IP) address and Transmission Control Protocol (TCP) state information, the device comprising:

To the extent that the preamble is limiting, Erickson in view of Tanenbaum96 discloses this limitation.

The host computer of Erickson operates a stack of protocol layers for “normal streams processing” for “slow applications” for which the host performs conventional protocol processing:
Ex.1005, Erickson at Fig. 3 (annotated), *see also id.* at 4:52-61 (“A traditional slow application 306 uses normal streams processing 308 to send information to a pass-through driver 310. The pass-through driver 310 initializes the physical hardware registers 320 of the I/O device adapter 314 to subsequently transfer the information through the I/O device adapter 314 to the commodity interface 322.”); *see also* Ex.1003, Horst Decl. at Appendix A, 1.P.3.

Erickson also discloses a fast-path (for “fast applications”), in which the I/O device adapter performs some of the protocol processing.

With the present invention, fast user applications 302 and 304 directly use a setup driver 312 to initialize the physical hardware registers 320, then send the information directly through the I/O device adapter 314 to the commodity interface 322 via virtual hardware 316 and 318. Thus, the overhead of the normal streams processing 308 and pass-through driver 310 are eliminated with the use of the virtual hardware 316 and 318 of the present invention, and fast applications 302 and 304 are able to send and receive information more quickly than slow application 306.

Ex.1005, Erickson at 4:61-5:5.

Erickson discloses that a user process on the host computer creates a *context for communication* by (1) opening a device driver and specifying the protocol type (e.g. UDP or TCP), source port number or address, whether the connection is synchronous or asynchronous, and setting up memory mapped registers, an
endpoint table and endpoint protocol data used for a protocol-specific script, id. at 6:1-9, 8:2-9; and (2) pre-negotiating connection details including a template header. Id. at 6:57-7:4.

Erickson describes this context as including “almost everything” concerning a UDP datagram “except the actual user data”:

Each user process has basically pre-negotiated almost everything about the datagram 602, except the actual user data 610. This means most of the fields in the three header areas 604, 606, and 608 are predetermined. In this example, the user process and the device driver has pre-negotiated the following fields from FIG. 6: (1) Ethernet Header 604 (Target Ethernet Address, Source Ethernet Address, and Protocol Type) … (2) IP Header 606 (Version, IP header Length, Service Type, Flag, Fragment Offset, Time_to_Live, IP Protocol, IP Address of Source, and IP Address of Destination); and (3) UDP Header 608 (Source Port and Destination Port). Only the shaded fields in FIG. 6, and the user data 610, need to be changed on a per-datagram basis.

Id. at 6:57-7:4.
Id. at Fig. 6.

As described above and shown in Figs. 6 and 7, Erickson’s pre-negotiated context includes a MAC layer address, IP address and UDP address. See id. at Figs. 6 and 7. Erickson thus teaches that “the context including a media access control (MAC) layer address, an Internet Protocol (IP) address.”
While Erickson’s exemplary context is UDP over IP, Erickson also discloses the fast path approach for TCP/IP, and refers the reader to Tanenbaum’s 1981 edition for details of TCP/IP. *Id.* at 4:38-43. A POSA would readily understand how to implement Erickson’s TCP/IP fast path in view of Tanenbaum96. *See* Section 9.3 (motivations to combine).

UDP is a connectionless protocol, and does not require setup or teardown of connections. *See* Section 4.2. Accordingly, the fast path can be used for all packets. Tanenbaum96 teaches that for TCP, only connections in the ESTABLISHED state should be processed on the fast path:

The key to fast TPDU [i.e. packet] processing is to separate out the normal case (one-way data transfer) and handle it specially. Although a sequence of special TPDUs are needed to get into the ESTABLISHED state, once there, TPDU processing is straightforward until one side starts to close the connection.

Let us begin by examining the sending side in the ESTABLISHED state when there are data to be transmitted…. The first thing the transport entity does is make a test to see if this is the normal case: the state is ESTABLISHED, neither side is trying to close the connection, a regular (i.e., not an out-of-band) full TPDU is being sent, and there is enough window space available at the receiver. If all conditions are met, no further tests are needed and the fast path through the sending transport entity can be taken.

Ex.1006, Tanenbaum96 at .583. Tanenbaum96 thus teaches that the sequence of
special packets needed to get into the ESTABLISHED state are handled on the conventional (slow) path.

A POSA would understand that, in view of Tanenbaum96, Erickson’s host operates the stack of protocol processing layers to create TCP connection using Erickson’s slow path. Once the connection is in the ESTABLISHED state, the host uses fast path TCP communication. Ex.1003, Horst Decl. at ¶¶33-35, 64 n.4, Appendix A, 1.P.3.

A POSA would readily understand how to modify Erickson’s UDP/IP fast path context to support TCP/IP based on the TCP/IP prototype header disclosed in Tanenbaum96, namely, the TCP state information is maintained in Erickson’s device (like the pre-negotiated information) for creating packets and performing Tanenbaum96’s bypass test:

References to what a POSA would understand in regards to the combination of Erickson and Tannenbaum96 are also supported by corresponding motivation for the combination. See Section 9.3 (motivations to combine).
Specifically, a POSA would also understand how to modify Erickson’s UDP/IP fast path context to include the TCP connection records described in Tanenbaum96:

Now let us look at fast path processing on the receiving side… For TCP, the connection record can be stored in a hash table for which some simple function of the two IP addresses and two ports is the key. Once the connection record has been located, both addresses and both ports must be compared to verify that the correct record has been found…. the TPDU [Transport Protocol Data Unit, i.e. packet] is then checked to see if it is a normal one: the state is ESTABLISHED, neither side is trying to close the connection, the TPDU is a full one,
no special flags are set, and the sequence number is the one expected. These tests take just a handful of instructions. If all conditions are met, a special fast path TCP procedure is called.

The fast path updates the connection record and copies the data to the user. While it is copying, it also computes the checksum, eliminating an extra pass over the data. If the checksum is correct, the connection record is updated and an acknowledgement is sent back. The general scheme of first making a quick check to see if the header is what is expected, and having a special procedure to handle that case, is called **header prediction.** Many TCP implementations use it.

Ex.1006, Tanenbaum96 at .584-.585 (underlining added, bold in original); see Ex.1003, Horst Decl. ¶¶143-47, Appendix A, 1.P.3.

The “connection records” disclosed in Tanenbaum96 are used to maintain TCP state:

When an application on the client machine issues a CONNECT request, the local TCP entity creates a connection record, marks it as being in the *SYN SENT* state, and sends a *SYN* segment. Note that many connections may be open (or being opened) at the same time on behalf of multiple applications, so the state is per connection and recorded in the connection record.

Ex.1006, Tanenbaum96 at .549 (underlining added). Erickson in view of Tanenbaum96 thus teaches “the context including a media access control (MAC)

-61-
layer address, an Internet Protocol (IP) address and Transmission Control Protocol (TCP) state information.”

To the extent that Erickson does not expressly disclose the host operating the stack of protocol layers to pre-negotiate the connection record, it would be obvious in view of Tanenbaum96. Tannenbaum teaches that a TCP connection must be in the ESTABLISHED state for fast path processing. Ex.1006, Tanenbaum96 at .584-.585. Further, Tanenbaum96 teaches that checking for established connections requires only a handful of instructions, and that packet processing for established connections is straightforward until one side starts to close the connection. This simple check to reduce processing complexity would motivate a POSA to follow Tanenbaum96’s fast path receive teaching. Id.at .583. See Section 9.3 (motivations to combine). Therefore, to the extent that Erickson does not expressly disclose the host operating a stack of protocol layers to create a context, it would be obvious in view of Tanenbaum96. See also Ex.1003, Horst Decl. at Appendix A, 1.P.3.

Accordingly, Erickson in view of Tanenbaum96 discloses that the first processor of the host computer operates a stack of protocol processing layers (its slow path stack for normal stream processing to set up a connection) that create a context for communication (registers 508 and 504, endpoint table 514, and endpoint protocol data 518, TCP protocol script and the pre-negotiated protocol
information), the context including a media access control (MAC) layer address, an Internet Protocol (IP) address and Transmission Control Protocol (TCP) state information (address information to send the TCP/IP packet over the network that Erickson pre-negotiates, as well as sequence number and TCP fields such as window size).

10.1.4. [1.1] a communication processing mechanism connected to the first processor,

Erickson discloses a communication processing mechanism connected to the first processor.

Specifically, Erickson’s I/O adapter (the “device”) includes a communication processing mechanism as shown in the below annotated figure, which includes a second processor (see below limitation [1.2]) with scripts that executes protocol scripts (1) to send data to the second apparatus via the commodity interface, and (2) receive data from the second apparatus, and transfer it to the correct host endpoint applications.

The pass-through driver 310 initializes the physical hardware registers 320 of the I/O device adapter 314 to subsequently transfer the information through the I/O device adapter 314 to the commodity interface 322. FIG. 4 is a block diagram describing a direct application interface (DAI) and routing of data between processes and an external data connection which is compatible with the present invention. Processes 402 and 404 transmit and receive information
directly to and from an interconnect 410 (e.g., I/O device adapter) through the DAI interface 408.

Ex.1005, Erickson at 4:58-5:10, see also id. at 4:18-23 (running scripts).

The “communication processing mechanism” of the I/O device adapter is connected to the first processor through standard computer I/O buses.

FIG. 2 is a block diagram illustrating a virtual hardware memory organization compatible with the present invention. I/O device adapters on standard I/O buses, such as ISA, EISA, MCA, or PCI buses….

Id. at 3:36-40.

These components are shown below:

Ex.1005, Erickson at Fig. 3 (annotated).
Accordingly, Erickson discloses a communication processing mechanism (the processor of the I/O device adapter with scripts to execute) connected to the first processor (via buses); see also Horst Decl. at Appendix A, 1.1.

10.1.5. [1.2] said communication processing mechanism containing a second processor

Erickson discloses the said communication processing mechanism containing a second processor.

Specifically, the I/O device adapter discloses a second processor because it discloses that the I/O device adapter executes “scripts” (program code).

A script is prepared by the operating system for the I/O device adapter to execute each time the specific user process programs its specific virtual hardware.

Ex.1005, Erickson at 4:18-23, see also id. at 7:48-8:26 (example script that, inter alia, executes arithmetic functions to calculate a checksum). A POSA would have thus understood Erickson to disclose a second processor (and certainly found it obvious that the I/O device adapter includes a processor). See Ex.1003, Horst Decl. at Appendix A, 1.2 (describing type of scripts and that a processor executes
Accordingly, Erickson discloses, and at least renders obvious, *said communication processing mechanism* (processor and scripts of I/O device adapter) *containing a second processor* (the processor of the I/O device adapter to execute the scripts).

**10.1.6.** [1.3] [second processor] running instructions to process a message packet such that the context is employed to transfer data contained in said packet to the first apparatus memory and Erickson in view of Tanenbaum96 discloses this limitation.

First, Erickson discloses running scripts, which requires running instructions. Ex.1005, Erickson at 4:18-23 (*A script is prepared by the operating system for the I/O device adapter to execute each time the specific user process programs its specific virtual hardware.*), *see also id.* at 7:48-8:26 (example script).

Erickson’s protocol scripts include *instructions to process a message packet such that the context is employed to transfer data contained in said packet to the first apparatus memory* by employing the context (present in, e.g., registers 504 and 508, endpoint table 514, and endpoint protocol data 518 and protocol script),

**10** Note that very definition of a processor is that it interprets and executes commands, i.e., the scripts of Erickson. Computer Dictionary, Microsoft (1994) (Ex.1037) at .010, .011.
to transfer incoming data “from the memory 512 of the I/O device adapter to the portions of main memory 502 associated with a process”:

The I/O device adapter implementation includes a software register 508 and a physical address buffer map 510 in the adapter's memory 512. An endpoint table 514 in the memory 512 is used to organize multiple memory pages for individual user processes. Each entry within the endpoint table 514 points to various protocol data 518 in the memory 512 in order to accommodate multiple communication protocols, as well as previously defined protocol scripts 516 in the memory 512, which indicate how data or information is to be transferred from the memory 512 of the I/O device adapter to the portions of main memory 502 associated with a user process.

_Id._ at 5:53-67.

Erickson discloses using this context to transfer the data to the host memory, i.e., process 402 or 404, via a DAI interface:

FIG. 4 is a block diagram describing a direct application interface (DAI) and routing of data between processes and an external data connection which is compatible with the present invention. Processes 402 and 404 transmit and receive information directly to and from an interconnect 410 (e.g., I/O device adapter) through the DAI interface 408. The information coming from the interconnect 410 is routed directly to a process 402 or 404 by use of virtual hardware and registers, rather than using a traditional operating system interface 406.
Id. at 4:53-5:14, Fig. 4, see also id. at Fig. 3 (illustrating that I/O device adapter 314 sends data to applications 302 and 304 that reside within the memory of the host computer), see also id. at 6:1-41, 8:17-37 and Ex.1003, Horst Decl. at Appendix A, 1.3 (explaining direct writing to memory by I/O device).

Accordingly, Erickson in view of Tanenbaum96 discloses running instructions (scripts) to process a message packet such that the context (including registers 508 and 504, endpoint table 514, endpoint protocol data 518, protocol scripts and pre-negotiated information) is employed (identifying the protocol type of the packet and how to transfer the data to the main memory) to transfer data contained in said packet (TCP/IP packet in view of Tanenbaum96) to the first apparatus memory (memory of host computer).

10.1.7. [1.4] the TCP state information is updated by said second processor.”

Erickson in view of Tanenbaum96 discloses that the TCP state information is updated by said second processor.

Specifically, Tanenbaum96 discloses:

Now let us look at fast path processing on the receiving side of Fig. 6-49. Step 1 is locating the connection record for the incoming TPDU. For ATM, …
The “connection records” disclosed in Tanenbaum96 are used to maintain TCP state:

When an application on the client machine issues a CONNECT request, the local TCP entity creates a connection record, marks it as being in the SYN SENT state, and sends a SYN segment. Note that many connections may be open (or being opened) at the same time on behalf of multiple applications, so the state is per connection and recorded in the connection record.

_Id._ at .549.

This connection record includes the transmission control block (TCB) that corresponds to the connection. See _id._; see also Ex.1003, Horst Decl. ¶¶39-41, 143, Appendix A, 1.4 (describing updating of, for example, sequence number of the state information).

Accordingly, Erickson in view of Tanenbaum96 teaches the TCP state information is updated by said second processor.
10.2. Claim 2 is unpatentable as obvious over Erickson in view of Tanenbaum96

10.2.1. [2] The device of claim 1, wherein said communication processing mechanism includes a receive sequencer with directions to classify said packet, wherein said packet contains control information corresponding to the stack of protocol layers.

Erickson in view of Tanenbaum96 discloses this limitation.

Erickson teaches classifying packet by protocol type because each requires a different script. See Ex.1005, Erickson at 5:41-51.

Further, it would have been obvious to adapt Erickson to include a receive sequencer to distinguish between fast and slow path processing\(^\text{11}\), using Tanenbaum96’s fast path “header prediction.” See Section 9.3 (discussing motivations to combine); see also Ex.1003, Horst Decl. ¶145.

\(^{11}\) Note that Erickson discloses a slow path (Slow Application) and fast path (Fast Application) and thus teaches the concept of classifying packets for each path. See Ex.1005, Erickson at Fig. 3.
Tanenbaum96 discloses a receive sequencer (its transport entity) that receives sequences of packets and classifies the received packets for fast path processing using header prediction:\(^\text{12}\):

Ex.1006, Tanenbaum96 at .585; see also Horst Decl. at Appendix A, 2.1 (describing “transport entity” as a processor performing the header prediction).

Accordingly, adapting Erickson to include Tanenbaum’s “header prediction” discloses said communication processing mechanism includes a receive sequencer with directions to classify said packet as a fast path packet.

\(^\text{12}\) Tannenbaum96 teaches that a “transport entity,” which may reside in the network interface card, performs this test. Ex.1006, Tanenbaum96 at .497-.498.
Further, a received TCP/IP packet contains control information corresponding to the stack of protocol layers in the TCP and IP headers:

![TCP and IP headers diagram](image)

*Fig. 6-50. (a) TCP header. (b) IP header. In both cases, the shaded fields are taken from the prototype without change.*

*Id.* at .584.

For example, the TCP header includes a “sequence number” that controls the packet’s placement within the application data and “port” fields that control the communication flow, and the IP header has a VER field (version of IP packet that controls its processing) and address fields that also control the communication flow. *See Ex.1003, Horst Decl.* Horst Decl. ¶31, Appendix A, 2.1.

Accordingly, Erickson in view of Tanenbaum96 discloses that the *communication processing mechanism includes a receive sequencer* (Erickson’s I/O device adapter using Tanenbaum96’s header prediction to classify packets for fast path processing) *with directions to classify said packet* (fast versus slow path), *wherein said packet contains control information corresponding to the stack of protocol layers* (control information in TCP/IP headers).
10.3. Claim 3 is unpatentable as obvious over Erickson in view of Tanenbaum96

10.3.1. [3] The device of claim 1, wherein said communication processing mechanism includes a receive sequencer with directions to generate a summary of a second message packet received from the network, said second packet containing control information corresponding to the stack of protocol layers, and said instructions including an instruction to compare said summary with said context.

Erickson in view of Tanenbaum96 discloses this limitation.

Erickson in view of Tanenbaum96 discloses that the communication processing mechanism includes a receive sequencer with directions to generate a summary of a second message packet received from the network. As noted for claim 2, it would have been obvious to combine Erickson with Tanenbaum96’s header prediction teachings, including to adapt Erickson to include Tanenbaum96’s header prediction. See claim 2 above.

Further, Tanenbaum96 discloses that the receive sequencer (the “transport entity” and header prediction) produces a summary of the incoming packets (and thus a “second packet”); specifically, Tanenbaum96 discloses using a hash of the IP addresses and TCP ports and then using the summary (IP addresses and ports of the headers) to compare against the context (the connection record) to verify the correct record is found:
Erickson in view of Tanenbaum96 discloses that the second packet contains control information corresponding to the stack of protocol layers:

Id. at .584.

For example, the TCP header includes a “sequence number” that controls the packet’s placement within the application data and “port” fields that control the
communication flow, and the IP header has a VER field (version of IP packet that controls its processing) and address fields that also control the communication flow. *See* Ex.1003, Horst Decl. Horst Decl. ¶31, Appendix A, 2.1.

Erickson in view of Tanenbaum96 also discloses that the instructions include an instruction to compare said summary with said context. As shown above, Tanenbaum96 discloses comparing the summary (the IP addresses and ports) to the connection record (context) to verify there is a match, and thus the packet is a proper candidate for fast path processing (for the I/O device adapter protocol processing when applying this teaching to Erickson). Ex.1006, Tanenbaum96 at .584; *see* Ex.1003, Horst Decl. at Appendix A, 3.1.

Accordingly, Erickson in view of Tanenbaum96 discloses that the communication processing mechanism includes a receive sequencer (Erickson’s I/O device adapter with Tanenbaum96’s header prediction) with directions (instructions) to generate a summary of a second message packet received from the network (IP addresses and TCP ports), said second packet containing control information corresponding to the stack of protocol layers (control information in TCP and IP headers), and said instructions including an instruction to compare said summary with said context (comparing summary to connection record to verify a match).
10.4. Claim 4 is unpatentable as obvious over Erickson in view of Tanenbaum96

As set forth below, Erickson in view of Tanenbaum discloses the limitations of claim 4. Erickson discloses creating a header. As discussed in claim 1, Erickson in view of Tanenbaum96 discloses using Erickson’s I/O device adapter for a TCP/IP connection. Accordingly, Erickson in view of Tanenbaum96 discloses creating a TCP/IP header in the context of a TCP/IP connection. Finally, it would have been obvious to prepend the header onto the data portion of the packet.

10.4.1. [4.1] The device of claim 1, wherein said instructions include a first instruction to create a header corresponding to said context and having control information corresponding to several of the protocol processing layers,

Erickson in view of Tanenbaum96 discloses this limitation.

Specifically, Erickson discloses that the I/O device adapter uses the scripts (which include a first instruction) to create a header.

Protocol scripts typically serve two functions. The first function is to describe the protocol the software application is using. This includes but is not limited to how to locate an application endpoint, and how to fill in a protocol header template from the application specific data buffer. The second function is to define a particular set of instructions to be performed based upon the protocol type. Each type of protocol will have its own script. Types of protocols include, but are not limited to, TCP/IP, UDP/IP, BYNET lightweight datagrams,
deliberate shared memory, active message handler, SCSI, and File Channel.

Ex.1005, Erickson at 5:41-51, see also 6:1-9, 6:57-7:4; see also Ex.1003, Horst Decl. at Appendix A, 4.1 (describing how a “a first instruction” is met by the scripts).

Erickson further discloses that the user process “spanks” a register to trigger a script on the adapter that creates headers from the header template and then transmits the packet:

FIG. 7 is a block diagram illustrating a UDP datagram template 702 (without a user data area) residing in the I/O device adapter's memory. The user process provides the starting address and the length for the user data in its virtual address space, and then "spanks" a GO register to trigger the I/O device adapter's execution of a predetermined script. The I/O device adapter stores the user data provided by the user process in the I/O device adapter's memory, and then transmits the completed UDP datagram 702 over the media.

Id. at 7:39-47.

Tanenbaum96 discloses creating TCP layer and IP layer headers:
As an example of how this principle works in practice, let us consider TCP/IP. Fig. 6-50(a) shows the TCP header. The fields that are the same between consecutive TPDU's on a one-way flow are shaded. All the sending transport entity has to do is copy the five words from the prototype header into the output buffer, fill in the next sequence number (by copying it from a word in memory), compute the checksum, and increment the sequence number in memory. It can then hand the header and data to a special IP procedure for sending a regular, maximum TPDU. IP then copies its five-word prototype header [see Fig. 6-50(b)] into the buffer, fills in the Identification field, and computes its checksum. The packet is now ready for transmission.

![Figure 6-50](image)

Ex.1006, Tanenbaum96 at .584.

Applying Tanenbaum96’s teachings to Erickson, the I/O device adapter of Erickson would produce TCP/IP headers (from the TCP/IP context that Erickson pre-negotiates, see Ex.1005, Erickson at 6:1-9, 6:57-7:4). See Ex.1003, Horst Decl. at ¶¶141, 143-44, Appendix A, 4.1

Note that the header includes control information:

![Figure 6-50](image)

Ex.1006, Tanenbaum96 at .584.
For example, the TCP header includes a “sequence number” that controls the packet’s placement within the application data and “port” fields that control the communication flow, and the IP header has a VER field (version of IP packet that controls its processing) and address fields that also control the communication flow. *See* Ex.1003, Horst Decl. at Appendix A, 4.1

Accordingly, Erickson in view of Tanenbaum96 discloses that the instructions (scripts) *include a first instruction to create a header corresponding to said context* (using header templates) and *having control information corresponding to several of the protocol processing layers* (information in the TCP/IP headers).

10.4.2. [4.2] and said instructions include a second instruction to prepend said header to second data for transmission of a second packet.

Erickson in view of Tanenbaum96 discloses this limitation.

Under the broadest reasonable construction standard, “prepend” would have been understood to mean “adds to the front.” *See* Section 7.2.1.

First, Erickson discloses a script (which includes “a second instruction”) for filling in a protocol header template.

Protocol scripts typically serve two functions. The first function is to describe the protocol the software application is using. This includes but is not limited to how to locate an application endpoint, and how to
fill in a protocol header template from the application specific data buffer.

Ex.1005, Erickson at 5:41-51; see also Ex.1003, Horst Decl. at Appendix A, 4.1 (describing how a “a second instruction” is met by the scripts).

As noted above (element 4.1), it would have been obvious to create a TCP/IP header in view of Tanenbaum96’s teachings.

It would have been obvious to add the headers to the front of data for transmission. There are at least two obvious approaches: prepending headers to data, or appending data to headers. Each approach is predictable and easy to implement, and a POSA would have been motivated to prepend the headers due to the header only be ready to add after further calculations (while the data is already present in the I/O device). See Ex.1003, Horst Decl. at Appendix A, 4.2

Accordingly, Erickson in view of Tanenbaum96 discloses that the instructions include a second instruction to prepend said header (via the Erickson header template) to second data for transmission of a second packet (the I/O device adapter sending the second packet onto the network).
10.5. Claim 5 is unpatentable as obvious over Erickson in view of Tanenbaum96

10.5.1. [5.1] The device of claim 1, wherein said communication processing mechanism has a direct memory access unit to send, based upon said context, said data from said communication processing mechanism to the first apparatus memory,

Erickson in view of Tanenbaum96 discloses this limitation.

Erickson discloses sending data to the memory of the first apparatus (the memory of the host computer) based on the context (resident in registers 504 and 508, endpoint table 514, and endpoint protocol data 518):

FIG. 5 is a block diagram illustrating the system organization between a main memory and an I/O device adapter memory which is compatible with the present invention. … An endpoint table 514 in the memory 512 is used to organize multiple memory pages for individual user processes. Each entry within the endpoint table 514 points to various protocol data 518 in the memory 512 in order to accommodate multiple communication protocols, as well as previously defined protocol scripts 516 in the memory 512, which indicate how data or information is to be transferred from the memory 512 of the I/O device adapter to the portions of main memory 502 associated with a user process.

Ex.1005, Erickson at 5:53-67, see also id. at 6:1-10 (describing that the user processes set up the context).
Erickson further depicts the I/O device adapter receiving data and directly providing it to an application (via memory of the host computer) in Figure 4:

**FIG. 4**

Processes 402 and 404 transmit and receive information directly to and from an interconnect 410 (e.g., I/O device adapter) through the DAI interface 408. The information coming from the interconnect 410 is routed directly to a process 402 or 404 by use of virtual hardware and registers, rather than using a traditional operating system interface 406.

*Id.* at Fig. 4 and at 5:5-5:14, *see also* Fig. 3 (illustrating that I/O device adapter 314 sends data to applications 302 and 304 that reside within the memory of the host computer).

Similarly, Tanenbaum discloses that the “fast path…copies the data to the user.” *Ex.1006, Tanenbaum96* at .585.
To accomplish direct data transfers, Erickson specifically describes a direct memory access (DMA) component of the I/O device adapter for transferring data from the host to the I/O device adapter.

Instead, the adapter would most likely retrieve the needed user data from the user process' virtual address space using direct memory access (DMA) into the main memory over the bus and retrieving the user data into some portion of the adapter's memory, where it could be referenced more efficiently.

Ex.1005, Erickson at 8:17-37.

In view of these disclosures, it would have been understood by a POSA, and certainly obvious to POSA, that the function of directly sending data from the I/O device adapter to the memory of the host computer would also occur via DMA. See Ex.1003, Horst Decl. at Appendix A, 5.1; see also Ex.1005, Erickson at Fig. 5 (direct connection from I/O device adapter memory 510 to host memory buffer pool 506). DMA was a well-known method to efficiently accomplish such transfers. Ex.1003, Horst Decl. at Sections V.H.1. and Appendix A, 5.1 (citing Ex.1038, U.S. Pat. No. 4,831,523, at 9:2-7 for support).

Accordingly, Erickson in view of Tanenbaum96 teaches that the communication processing mechanism has a direct memory access unit (Erickson’s DMA) to send, based upon said context (e.g., information in registers 508, endpoint table 514, and protocol data 518), said data (received packets) from
said communication processing mechanism to the first apparatus memory (host memory of corresponding user processes).

10.5.2. [5.2] without a header accompanying said data.

Erickson in view of Tanenbaum96 discloses removing the headers before sending the data to host memory.

It would have been understood by a POSA, and certainly obvious, that Erickson strips the headers off the packets before sending the data to the host memory because it sends the data directly to the user process and skips the normal stack processing. See Ex.1003, Horst Decl. at Appendix A, 5.2; Ex.1005, Erickson at 4:53-5:14, 5:53-67, 5:53-67, Fig. 4 (sending data directly to host memory). Specifically, as the I/O device adapter is performing the TCP/IP processing, it would be understood that it strips the TCP/IP, etc., headers of the data before sending it. See Ex.1003, Horst Decl. at Appendix A. 5.2.

To the extent that Erickson does not expressly disclose stripping off these headers, it would be obvious in view of Tanenbaum96. Tanenbaum96 describes the fast path “copying” the data to the user,” i.e., the “data” and not the header. Ex.1006, Tanenbaum96 at .585.\(^\text{13}\) The reason for offloading this processing is so

\(^{13}\) Recall Tanenbaum96 discloses that the transport entity, which performs these functions, may reside in the network interface card. Ex.1006, Tanenbaum96 at
that the host does not perform these functions (moreover, the user application is expecting only data, not data with headers, because it only receives data after protocol processing). See Ex.1003, Horst Decl. Appendix A, 5.2.

10.6. Claim 6 is unpatentable as obvious over Erickson in view of Tanenbaum96

10.6.1. [6] The device of claim 1, wherein said context includes a receive window of space in the memory that is available to store application data, and said communication processing mechanism advertises said receive window.

Erickson in view of Tanenbaum96 discloses this limitation.

As noted, it would be obvious to combine Erickson with Tanenbaum96’s TCP/IP teachings to effectuate a TCP/IP connection with Erickson’s I/O device adapter. See Section 9.3 (describing motivations to combine). The TCP/IP headers, as Tanenbaum96 teaches, include a “window size,” which is part of the context:

---

.497-.498. In view of Erickson, the I/O device adapter (part of the network interface card) would be performing these functions, and thus sending data from the I/O device adapter to the host. See Ex.1003, Horst Decl. Appendix A, 5.2.

14 The receive window is part of the TCP headers, so it would be understood as part of the connection context, otherwise the I/O device adapter of Erickson could
Ex.1006, Tanenbaum96 at .584.

The window size is how much space in the receiver’s memory is available for received data (e.g. buffer space), and the communication processing mechanism advertises the receive window by including it in each packet: “TCP Congestion Control … TCP attempt to achieve this goal by dynamically manipulating the window size. … The receiver can specify a widow based on its buffer size.” Ex.1006, Tanenbaum96 at .554-555; see also Horst Decl. at Section V.B.9 and Appendix A, 6.1.

Accordingly, Erickson in view of Tanenbaum96 discloses wherein said context includes a receive window of space in the memory that is available to store application data (window size field of the TCP/IP packet), and said not construct packets as it teaches. Ex.1005, Erickson at 5:41-51; see also Ex.1003, Horst Decl. Appendix A, 6.1.
communication processing mechanism advertises said receive window (via sending
the packets which include this information).

10.7. Claim 7 is unpatentable as obvious over Erickson in view of Tanenbaum96

10.7.1. [7] The device of claim 1, wherein said context includes TCP ports of said first and said second
apparatuses.

Erickson in view of Tanenbaum96 discloses this limitation.

First, Erickson discloses pre-negotiating ports for datagrams to the I/O
device adapter as part of the connection setup. Ex.1005, Erickson at 6:57-7:4 (“In
this example, the user process and the device driver has pre-negotiated the
following fields from FIG. 6: … (3) UDP Header 608 (Source Port and Destination
Port).” See also id. at Fig. 6.

Second, as noted, it would be obvious to combine Erickson with
Tanenbaum96’s teachings for a TCP/IP connection. See Section 9.3 (describing
motivations to combine). A TCP packet includes a TCP source and destination
port number, and thus the Erickson’s pre-negotiating for a TCP/IP connection in
view of Tanenbaum96 would include creating these values as part of the context15:

15 Again, the I/O device adapter of Erickson would be understood to store this
information to create headers. Ex.1005, Erickson at 5:41-51. Note that this
Accordingly, Erickson in view of Tanenbaum96 discloses that *said context includes TCP ports of said first and said second apparatuses* (the source and destination ports).

---

information is part of the standard TCB that devices store for a TCP/IP connection.

Ex.1003, Horst Decl. Appendix A, 7.1.
11. CONCLUSION

For the reasons set forth above, *inter partes* review of claims 1-7 of the 036 Patent is requested.
Respectfully submitted,

WEIL, GOTSHAL AND MANGES LLP

Dated: May 9, 2017

By: /s/ Garland T. Stephens

Garland T. Stephens
Registration No. 37,242
(garland.stephens@weil.com)

Postal & Hand-Delivery Address:
Weil, Gotshal & Manges LLP
700 Louisiana St., Suite 1700
Houston, TX 77002
T: 713-546-5011
F: 713-224-9511
CERTIFICATE OF COMPLIANCE

Pursuant to 37 C.F.R. § 42.24 et seq., the undersigned certifies that this document complies with the type-volume limitations. This document contains 13,364 words as calculated by the “Word Count” feature of Microsoft Word 2010, the word processing program used to create it.

WEIL, GOTSHAL AND MANGES LLP

Dated: May 9, 2017

By: /s/ Garland T. Stephens

Garland T. Stephens
Registration No. 37,242
(garland.stephens@weil.com)

Postal & Hand-Delivery Address:
Weil, Gotshal & Manges LLP
700 Louisiana St., Suite 1700
Houston, TX 77002
T: 713-546-5011
F: 713-224-951
CERTIFICATE OF SERVICE

The undersigned certifies, in accordance with 37 C.F.R. § 42.105 and § 42.6(e), that service was made on the Patent Owner as detailed below.

Date of service: May 9, 2017
Manner of service: EXPRESS MAIL
Documents served: Petition for Inter Partes Review of U.S. Pat. No. 7,237,036 with List
Power of Attorney
Exhibits Ex.1001 through Ex.1039

Persons served: Patent Owner’s Address of Record:
MARK LAUER
Silicon Edge Law Group LLP
7901 STONERIDGE DRIVE
SUITE 528
Pleasanton, California 94588
Fax: (925)-621-2119
Phone: 925-621-2121
Email: Mark@SiliconEdgeLaw.com

Additional Addresses Known as Likely to Effect Service:
Claude M Stern
Quinn Emanuel Urquhart & Sullivan - Redwood
555 Twin Dolphin Dr.
5th Floor
Redwood Shores, CA 94065
Phone: 650-801-5002
Fax: 650-801-5100
Email: claudestern@quinnemanuel.com
Petition for *Inter Partes* Review of U.S. Patent No. 7,237,036

Dated: May 9, 2017

/s/ Garland T. Stephens
Garland T. Stephens
Lead Counsel for Petitioners
Registration No. 37,242