UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

CAVIUM, INC.

Petitioner

v.

ALACRITECH, INC.

Patent Owner

Case IPR. No. **Unassigned**

U.S. Patent No. 8,805,948

Title: INTELLIGENT NETWORK INTERFACE SYSTEM AND METHOD FOR PROTOCOL PROCESSING

Petition For *Inter Partes* Review of U.S. Patent No. 8,805,948 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

Page

Exhibit List

Exhibit #	Description
Ex.1001	U.S. Patent No. 8,805,948 ("948 Patent")
Ex.1002	Excerpts from Prosecution File History of U.S. Patent No. 8,805,948 ("948 File History")
Ex.1003	Declaration of Robert Horst
Ex.1004	*Curriculum Vitae* of Robert Horst
Ex.1005	U.S. Patent No. 5,768,618 ("Erickson")
Ex.1006	Tanenbaum, Andrew S., *Computer Networks*, Prentice-Hall, Inc., New Jersey (1996). ("Tanenbaum96")
Ex.1007	Transmission Control Protocol, "Darpa Internet Protocol Specification", RFC: 793, Sept. 1981. ("RFC 793")
Ex.1008	Stevens, W. Richard, *TCP/IP Illustrated Volume 1: The Protocols*, Addison-Wesley (1994). ("Stevens1")
Ex.1009	Lilinkamp, J., Mandell. R. and Padlipsky, M., "Proposed Host-Front End Protocol", Network Working Group Request for Comments: 929, Dec. 1984. ("RFC 929")
Ex.1010	*Not Used*
Ex.1011	Librarian Declaration of Rice Mayors regarding Andrew S. Tanenbaum, Computer Networks (3rd ed. 1996) (Ex.1006, "Tanenbaum96")
Ex.1012	*Not Used*
Ex.1013	Stevens, W. Richard and Gary R. Wright, *TCP/IP Illustrated Volume 2: The Implementation*, Addison-Wesley (1995). ("Stevens2")
Ex.1014	*Not Used*

Exhibit #	Description
Ex.1015	Thia, Y.H., Woodside, C.M., "A Reduced Operation Protocol Engine (ROPE) for a Multiple-Layer Bypass Architecture", Protocols for High Speed Networks (Dordrecht), 1995. ("Thia and Woodside")
Ex.1016	Biersack, E. W., Rütsche E., "Demultiplexing on the ATM Adapter: Experiments with Internet Protocols in User Space", Journal on High Speed Networks, Vol. 5, No. 2, May 1996. ("Biersack")
Ex.1017	Rütsche, E., Kaiserswerth, M., "TCP/IP on the Parallel Protocol Engine", Proceedings, IFIP Conference on High Performance Networking, Liege (Belgium), Dec. 1992. ("Rütsche92")
Ex.1018	Rütsche, E., "The Architecture of a Gb/s Multimedia Protocol Adapter", Computer Communication Review, 1993. ("Rütsche93")
Ex.1019	Padlipsky, M. A., "A Proposed Protocol for Connecting Host Computers to Arpa-Like Networks Via Directly-Connected Front End Processors", Network Working Group RFC #647, Nov. 1974. ("RFC 647")
Ex.1020	U.S. Patent No. 5,619,650 ("Bach")
Ex.1021	U.S. Patent No. 5,915,124 ("Morris")
Ex.1022	Cooper, E.C., et al., "Protocol Implementation on the Nectar Communication Processor", School of Computer Science, Carnegie Mellon University, Sept. 1990. ("Cooper")
Ex.1023	Kung, H.T., et al., "A Host Interface Architecture for High-Speed Networks", School of Computer Science, Carnegie Mellon University and Network Systems Corporation. ("Kung")
Ex.1024	Exhibit D to Declaration of Dr. Gregory L. Chesson in Support of Microsoft's Opposition to Alacritech's Motion for Preliminary Injunction: "Protocol Engine Handbook", Protocol Engines Incorporated, Oct. 1990. ("Chesson")

Exhibit #	Description
Ex.1025	Kanakia, H., Cheriton, D.R., "The VMP Network Adapter Board (NAB): High-Performance Network Communication for Multiprocessors", Communications Architectures & Protocols, Stanford University, Aug. 1988. ("Kanakia")
Ex.1026	Kung, H.T., Cooper, E.C., et al., "Network-Based Multicomputers: An Emerging Parallel Architectures", School of Computer Science, Carnegie Mellon University. ("Kung and Cooper")
Ex.1027	Dalton, C., et al., "Afterburner: Architectural Support for High-Performance Protocols", Networks & Communications Laboratories, HP Laboratories Bristol, July 1993. ("Dalton")
Ex.1028	Murphy, E., Hayes, S., Enders, M., *TCP/IP Tutorial and Technical Overview Fifth Edition*, Prentice-Hall, Inc. New Jersey, (1995). ("Murphy")
Ex.1029	MacLean, A.R., Barvick, S. E., "An Outboard Processor for High Performance Implementation of Transport Layer Protocols", IEEE Globecom '91, Phoenix, AZ, Dec. 1991. ("MacLean")
Ex.1030	Clark, D.D., et al., "An Analysis of TCP Processing Overhead", IEEE Communications Magazine, June 1989. ("Clark")
Ex.1031	Provisional Application 60-061,809 ("Alacritech 1997 Provisional Application")
Ex.1032	Culler, E.C., et al., "Parallel Computing on the Berkeley NOW", Computer Science Division, University of California, Berkeley. ("Culler")
Ex.1033	"Gigabit Ethernet Technical Brief: Achieving End-to-End Performance", Alteon Networks, Inc. First Edition, Sept. 1996. ("Alteon")
Ex.1034	Smith, J.A., Primmer, M., "Tachyon: A Gigabit Fibre Channel Protocol Chip", Hewlett-Packard Journal, Article 12, Oc. 1996. ("Smith")

Exhibit #	Description
Ex.1035	Patterson, D.A., Hennessy, J.L., *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Publishers, Inc., San Mateo, CA (1990). ("Patterson")
Ex.1036	Internet Protocol, "Darpa Internet Protocol Specification", RFC: 791, Sept. 1981. ("RFC 791")
Ex.1037	*Not Used*
Ex.1038	Woodside, C. M., K. Ravindran, and R. G. Franks. "The protocol bypass concept for high speed OSI data transfer." IFIP Workshop on Protocols for High Speed Networks. 1990. ("Woodside")
Exs.1039-1062	*Not Used*
Ex.1063	Librarian Declaration of Pamela Stansbury regarding Gary R. Wright & W. Richard Stevens, TCP/IP Illustrated: The Implementation (1995) (Ex.1013, "Stevens2").
Ex.1064	Librarian Declaration of Lisa Rowlison de Ortiz re Y.H. Thia & C.M. Woodside, A Reduced Operation Protocol Engine (ROPE) for a Multiple-layer Bypass Architecture (1995) (Ex.1015, "Thia")

1. INTRODUCTION

Pursuant to 35 U.S.C. §§ 311-319 and 37 C.F.R. §§ 42.1-.80, 42.100-.123, Cavium, Inc. ("Petitioner" or "Cavium") hereby petitions the Patent Trial and Appeal Board to institute an *inter partes* review of claims 1, 3, 6-9, 11, 14-17, 19, and 21-22 of U.S. Patent No. 8,805,948, titled "Intelligent Network Interface System and Method for Protocol Processing" (Ex.1001, the "948 Patent"), and cancel those claims as unpatentable.

2. REQUIREMENTS FOR PETITION FOR INTER PARTES REVIEW

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

Petitioner certifies that the 948 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 948 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.

Lead Counsel	**Back-Up Counsel**
Patrick D. McPherson Registration No. 46,255 (PDMcPherson@duanemorris.com) Postal & Hand-Delivery Address: Duane Morris LLP 505 9th Street, N.W., Suite 1000 Washington, DC 20004 T: (202) 776-5214; F: (202) 776-7801 *Attorney for Cavium, Inc.*	David T. Xue, Ph.D. Registration No. 54,554 (DTXue@duanemorris.com) Postal & Hand-Delivery Duane Morris LLP 2475 Hanover St. Palo Alto, CA 94304 T: (650) 847-4153; F: (650) 444-0459 *Attorneys for Cavium, Inc.*

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

Petitioner, Cavium, Inc. is the real-party-in-interest. No other parties exercised or could have exercised control over this Petition; no other parties funded or directed this Petition. *See* Office Patent Trial Practice Guide, 77 Fed. Reg. 48759-60.

2.4. Notice of Related Matters (37 C.F.R. § 42.8(b)(2))[1]

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

[1] These petitions will be filed concurrently or within a few days.

addition to this Petition, Petitioner is filing petitions for U.S. Patent Nos. 7,237,036; 7,337,241; 7,673,072; 7,945,699; 8,131,880; 7,124,205; and 9,055,104.

The 948 Patent is subject to a pending request for IPR filed by Intel on May 9, 2017 (IPR-2017-01395).

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:

1. U.S. Patent Application No. 11/821,820 (filed Jun. 25, 2007, issued

Mar. 2, 2010 as 7,673,072 patent);

2. U.S. Patent Application No. 10/092,967 (filed Mar. 6, 2002, issued Jul. 8, 2003 as 6,591,302 patent);

3. U.S. Patent Application No. 10/023,240 (filed Dec. 17, 2001, issued Nov. 15, 2005 as 6,965,941 patent);

4. U.S. Patent Application No. 09/970,124 (filed Oct. 2, 2001, issued Oct. 17, 2006 as 7,124,205 patent);

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);

7. U.S. Patent Application No. 09/802,550 (filed Mar. 9, 2001, issued Dec. 2, 2003 as 6,658,480 patent);

8. U.S. Patent Application No. 09/802,551 (filed Mar. 9, 2001, issued Jul. 11, 2006 as 7,076,568 patent);

9. U.S. Patent Application No. 09/801,488 (filed Mar. 7, 2001, issued Feb. 3, 2004 as 6,687,758 patent);

10. U.S. Patent Application No. 09/789,366 (filed Feb. 20, 2001, issued Jun. 29, 2004 as 6,757,746 patent);

11. U.S. Patent Application No. 09/675,700 (filed Sept. 29, 2000, issued

Dec. 31, 2013 as 8,621,101 patent);

12. 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. 09/514,425 (filed Feb. 28, 2000, issued Jul. 30, 2002 as 6,427,171 patent);

14. U.S. Patent Application No. 09/464,283 (filed Dec. 15, 1999, issued Jul. 30, 2002 as 6,427,173 patent);

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

16. U.S. Patent Application No. 09/416,925 (filed Oct. 13, 1999, issued Oct. 22, 2002 as 6,470,415 patent);

17. U.S. Patent Application No. 09/384,792 (filed Aug. 27, 1999, issued Aug. 13, 2002 as 6,434,620 patent);

18. U.S. Patent Application No. 09/141,713 (filed Aug. 28, 1998, issued May 14, 2002 as 6,389,479 patent);

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

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

21. U.S. Patent Application No. 60/061,809 (expired).

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

1. U.S. Patent Application No. 10/092,967 (filed Mar. 6, 2002, issued Jul. 8, 2003 as 6,591,302 patent);

2. U.S. Patent Application No. 10/260,112 (filed Sept. 27, 2002, issued Jul. 26, 2007 as 7,237,036 patent);

3. U.S. Patent Application No. 10/261,051 (filed Sept. 30, 2002, issued Sept. 13, 2011 as 8,019,901 patent);

4. U.S. Patent Application No. 11/821,820 (filed Jun. 25, 2007, issued Mar. 2, 2010 as 7,673,072 patent);

5. U.S. Patent Application No. 11/582,199 (filed Oct. 16, 2006, issued Feb. 16, 2010 as 7,664,883 patent).

The patent family to which the 072 Patent belongs contains 19 additional U.S. patents:

1. U.S. Patent Application No. 10/260,112 (filed Sept. 27, 2002, issued Jul. 26, 2007 as 7,237,036 patent);

2. U.S. Patent Application No. 10/092,967 (filed Mar. 6, 2002, issued Jul. 8, 2003 as 6,591,302 patent);

3. U.S. Patent Application No. 10/023,240 (filed Dec. 17, 2001, issued Nov. 15, 2005 as 6,965,941 patent);

4. U.S. Patent Application No. 09/970,124 (filed Oct. 2, 2001, issued Oct. 17, 2006 as 7,124,205 patent);

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);

7. U.S. Patent Application No. 09/802,550 (filed Mar. 9, 2001, issued Dec. 2, 2003 as 6,658,480 patent);

8. U.S. Patent Application No. 09/802,551 (filed Mar. 9, 2001, issued Jul. 11, 2006 as 7,076,568 patent);

9. U.S. Patent Application No. 09/801,488 (filed Mar. 7, 2001, issued Feb. 3, 2004 as 6,687,758 patent);

10. U.S. Patent Application No. 09/789,366 (filed Feb. 20, 2001, issued Jun. 29, 2004 as 6,757,746 patent);

11. U.S. Patent Application No. 09/675,700 (filed Sept. 29, 2000, issued Dec. 31, 2013 as 8,621,101 patent);

12. 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. 09/514,425 (filed Feb. 28, 2000, issued Jul. 30, 2002 as 6,427,171 patent);

14. U.S. Patent Application No. 09/464,283 (filed Dec. 15, 1999, issued Jul. 30, 2002 as 6,427,173 patent);

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

16. U.S. Patent Application No. 09/416,925 (filed Oct. 13, 1999, issued Oct. 22, 2002 as 6,470,415 patent);

17. U.S. Patent Application No. 09/384,792 (filed Aug. 27, 1999, issued Aug. 13, 2002 as 6,434,620 patent);

18. U.S. Patent Application No. 09/141,713 (filed Aug. 28, 1998, issued May 14, 2002 as 6,389,479 patent);

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

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

21. U.S. Patent Application No. 60/061,809 (expired).

The patent family to which the 948 Patent belongs contains 2 additional U.S. patents:

1. U.S. Patent Application No. 09/692,561 (filed Oct. 18, 2000, issued Jan. 14, 2014 as 8,631,140 patent);

2. U.S. Patent Application No. 09/067,544 (filed Apr. 27, 1998, issued May 1,

2001 as 6,226,680 patent);

3. U.S. Patent Application No. 60/061,809 (expired).

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

1. U.S. Patent Application No. 10/881,271 (filed Jun. 29, 2004, issued Dec. 2, 2008 as 7,461,160 patent);

2. U.S. Patent Application No. 09/789,366 (filed Feb. 20, 2001, issued Jun. 29, 2004 as 6,757,746 patent);

3. U.S. Patent Application No. 09/748,936 (filed Dec. 26, 2000, issued Dec. 25, 2001 as 6,334,153 patent);

4. U.S. Patent Application No. 09/692,561 (filed Oct. 18, 2000, issued Jan. 14, 2014 as 8,631,140 patent);

5. U.S. Patent Application No. 09/675,484 (filed Sept. 29, 2000, issued Oct. 19, 2004 as 6,807,581 patent);

6. U.S. Patent Application No. 09/675,700 (filed Sept. 29, 2000, issued Dec. 31, 2013 as 8,621,101 patent);

7. U.S. Patent Application No. 09/514,425 (filed Feb. 28, 2000, issued Jul. 30, 2002 as 6,427,171 patent);

8. U.S. Patent Application No. 09/464,283 (filed Dec. 15, 1999, issued Jul. 30, 2002 as 6,427,173 patent);

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

10. U.S. Patent Application No. 09/416,925 (filed Oct. 13, 1999, issued Oct. 22, 2002 as 6,470,415 patent);

11. U.S. Patent Application No. 09/384,792 (filed Aug. 27, 1999, issued Aug. 13, 2002 as 6,434,620 patent);

12. U.S. Patent Application No. 09/141,713 (filed Aug. 28, 1998, issued May 14, 2002 as 6,389,479 patent);

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

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

15. U.S. Patent Application No. 60/061,809 (expired);

16. U.S. Patent Application No. 13/108,729 (filed May 16, 2011, issued Sept. 17, 2003 as 8,539,112 patent).

The patent family to which the 880 Patent belongs contains 6 additional U.S. patents:

1. U.S. Patent Application No. 10/005,536 (filed Nov. 7, 2001, issued Jan. 23, 2007 as 7,167,926 patent);

2. U.S. Patent Application No. 09/514,425 (filed Feb. 28, 2000, issued Jul. 30, 2002 as 6,427,171 patent);

3. U.S. Patent Application No. 09/464,283 (filed Dec. 15, 1999, issued Jul. 30, 2002 as 6,427,173 patent);

4. U.S. Patent Application No. 09/384,792 (filed Aug. 27, 1999, issued Aug. 13, 2002 as 6,434,620 patent);

5. U.S. Patent Application No. 09/141,713 (filed Aug. 28, 1998, issued May 14, 2002 as 6,389,479 patent);

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);

8. U.S. Patent Application No. 60/061,809 (expired).

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

1. U.S. Patent Application No. 10/413,256 (filed Apr. 22, 2003, issued Jun. 2, 2009 as 7,543,087 patent);

U.S. Patent Application No. 60/374,788 (expired).

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

1. U.S. Patent Application No. 10/092,967 (filed Mar. 6, 2002, issued Jul. 8, 2003 as 6,591,302 patent);

2. U.S. Patent Application No. 10/023,240 (filed Dec. 17, 2001, issued

Nov. 15, 2005 as 6,965,941 patent);

3. U.S. Patent Application No. 09/464,283 (filed Dec. 15, 1999, issued Jul. 30, 2002 as 6,427,173 patent);

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

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);

8. U.S. Patent Application No. 09/141,713 (filed Aug. 28, 1998, issued May 14, 2002 as 6,389,479 patent);

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

10. U.S. Patent Application No. 09/416,925 (filed Oct. 13, 1999, issued Oct. 22, 2002 as 6,470,415 patent);

11. U.S. Patent Application No. 09/514,425 (filed Feb. 28, 2000, issued Jul. 30, 2002 as 6,427,171 patent);

12. 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. 09/675,700 (filed Sept. 29, 2000, issued

Dec. 31, 2013 as 8,621,101 patent);

14. U.S. Patent Application No. 09/789,366 (filed Feb. 20, 2001, issued Jun. 29, 2004 as 6,757,746 patent);

15. U.S. Patent Application No. 09/801,488 (filed Mar. 7, 2001, issued Feb. 3, 2004 as 6,687,758 patent);

16. U.S. Patent Application No. 09/802,551 (filed Mar. 9, 2001, issued Jul. 11, 2006 as 7,076,568 patent);

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

18. U.S. Patent Application No. 09/802,550 (filed Mar. 9, 2001, issued Dec. 2, 2003 as 6,658,480 patent);

19. U.S. Patent Application No. 09/855,979 (filed May 14, 2001, issued Nov. 7, 2006 as 7,133,940 patent);

20. U.S. Patent Application No. 09/970,124 (filed Oct. 2, 2001, issued Oct. 17, 2006 as 7,124,205 patent);

21. U.S. Patent Application No. 09/464,283 (filed Dec. 15, 1999, issued Jul. 30, 2002 as 6,427,173 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. 04-1679.

2.6. Proof of Service

Proof of service of this Petition on the Patent Owner at the correspondence address of record for the 948 Patent is attached.

3. IDENTIFICATION OF CLAIMS BEING CHALLENGED (§42.104(B))

Ground #1: Claims 1, 3, 6-9, 11, 14-17, 19, and 21-22 of the 948 Patent are invalid under (pre-AIA) 35 U.S.C. § 103(a) on the ground that they are obvious over Y.H. Thia & C.M. Woodside, A Reduced Operation Protocol Engine (ROPE) for a Multiple-layer Bypass Architecture (1995) (Ex.1015, "Thia") in combination with Andrew S. Tanenbaum, Computer Networks (3rd ed. 1996) (Ex.1006, "Tanenbaum96") in further combination with 2 Gary R. Wright & W. Richard Stevens, TCP/IP Illustrated: The Implementation (1995) (Ex.1013, "Stevens2").

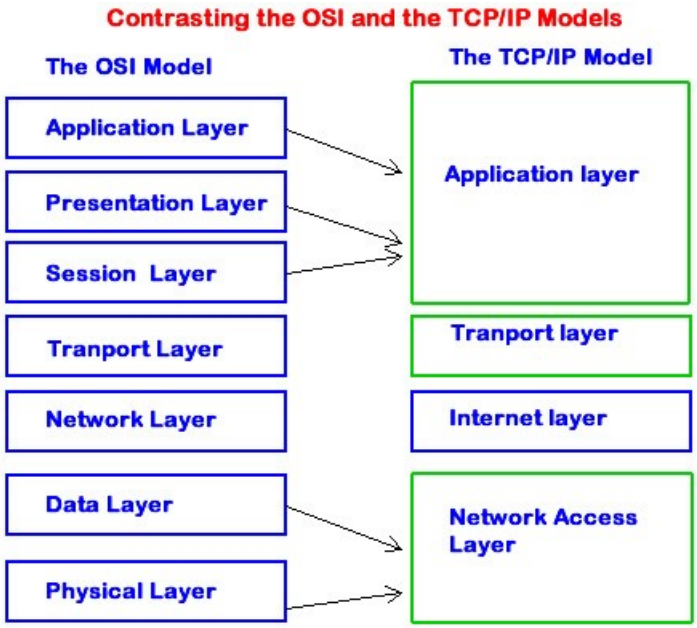
4. BACKGROUND OF TECHNOLOGY

This section provides a brief background on the technology at issue, focusing on layered network protocols generally, the TCP/IP protocol, offloading protocol processing from a host to a NIC, and Direct Memory Access ("DMA"). Petitioner's declarant Dr. Horst provides a tutorial on the technology and discusses the state of the art as well. Ex.1003, ¶¶21-88.

A. Layered Network Protocols

The seven-layer Open Systems Interconnect (OSI) model describes a logical network layering model including physical, data link, network, transport, session, presentation and application layers. *Id*. ¶23.

1. 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. There is a well-known correspondence between TCP/IP layers and the OSI model. The following figure shows the relationship between the OSI and TCP/IP layering.



Available at http://mitigationlog.com/how-tcpip-and-reference-osi-model-works/.

Conceptually, each layer is responsible only for its respective functions. Ex.1003, ¶25. This enables, for example, hiding the complexity of the physical data

connection (that is, actually transmitting and receiving the data onto the physical wires) from the data link, network and transport layers above. *Id.* Likewise, the lowest layer must transmit and receive the data on the physical wires, but need not worry about what application the data belongs to or whether the packets are received in order. *Id.*

B. TCP/IP

The 948 Patent relates to an intelligent network interface card that provides a "fast path" that avoids host protocol processing for most packets in a large multipacket message. Ex.1001, Abstract. The claims are all directed to TCP/IP.

By October 14, 1996, the critical date of the 948 Patent, TCP/IP was one of the most popular networking protocols, overtaking the OSI protocols. *See* Ex.1006, .016 ("The OSI protocols have quietly vanished, and the TCP/IP protocol suite has become dominant.") TCP/IP was standardized in a series of publically available Request for Comments (RFCs) published by the Internet Engineering Task Force, including RFC 793, entitled "Transmission Control Protocol" and RFC 791, entitled "Internet Protocol." Ex.1007; Ex.1036. Free implementations of TCP/IP were widely available and widely used. Ex.1003, ¶27.

TCP/IP consists of two parts: (1) Transmission Control Protocol (TCP), which provides virtual bi-directional connections that are 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 (IP packets) to any routable Internet address, without any reliability or ordering guarantees. IP also provides for fragmentation and reassembly. Fragmentation occurs when an IP packet must be divided ("fragmented") into smaller packets en route for transmission over an intermediate network with a small packet size. TCP/IP can be transmitted over a variety of physical media, such as Ethernet. *Id*. ¶28.

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 Media 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*. ¶29.

As shown in the figure below, in typical TCP/IP processing, the packet is built from the top down, i.e., each layer encapsulates what it receives from the above layer by concatenating an additional header associated with that layer. *Id.* ¶30. 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 above layer. *Id.* The user data without headers is eventually delivered to the relevant application.

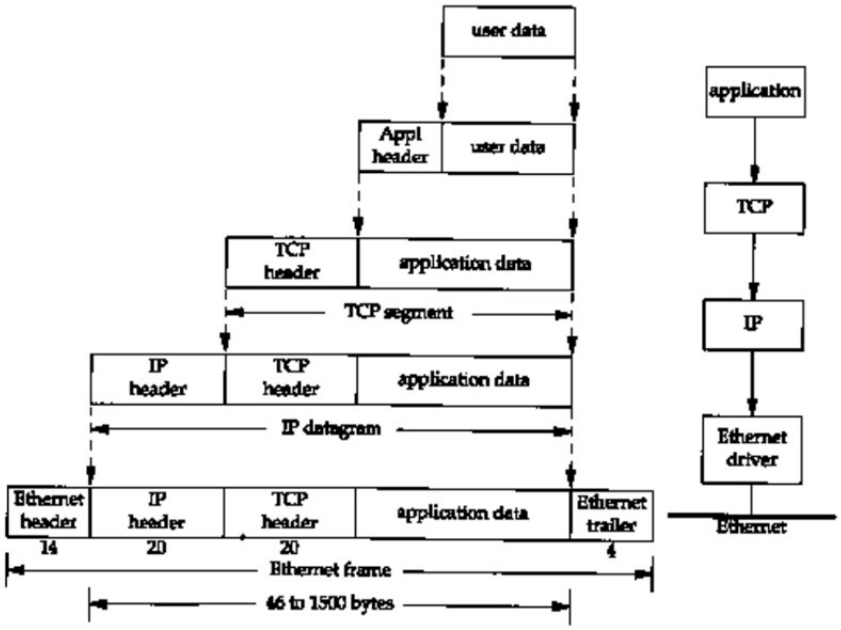


Figure 1.7 Encapsulation of data as it goes down the protocol stack.

Ex.1008, Fig. 1.7.

Starting from the lowest layer, the MAC (media access control) layer handles the actual transmission on the physical media. The header for this layer, e.g., an Ethernet header, includes a MAC address that is the address of the network interface (e.g., on a computer or router) on a local area network. Ex.1003, ¶31.

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. *Id.* ¶33. The IP header also has a flag that indicates whether the packet has been fragmented. *Id.*

Above the IP layer is the TCP (Transport) 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. *Id.* ¶34. The TCP layer tracks the sequence of packets, ensuring assembly of packets in the proper order by using sequence numbers in its headers. *Id.* ¶35. The transport services can be performed by software or the network interface card. *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 in the TCP packet headers. Ex.1003, ¶35.

TCP maintains a finite state machine to manage the state of each connection. Connections begin in a CLOSED state. A server can move from CLOSED into a LISTEN state, which waits passively for a client to open a connection by sending a special packet called a SYN packet. After the server acknowledges the SYN packet, the connection is in the ESTABLISHED state. Either side can close the connection by sending a special packet called a FIN packet. The TCP finite state machine and

associated messages are described in detail in RFC 793. RFC 793 describes the data structure for storing the information needed to maintain a TCP connection as a Transmission Control Block (TCB). Ex.1007, .016; Ex.1003, ¶36.

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.* ¶37.

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 the protocol stack has stripped the MAC, IP and TCP headers from the packet. *Id*. ¶38.

C. Protocol Offload and Fast-Path Processing

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, ¶40.

As early as 1974, front-end protocol offload (offloading protocol processing to a processor outside of the host) 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. Ex.1019; Ex.1003, ¶¶41-42.

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, .002; Ex.1003, ¶43. 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, .015-.016; Ex.1003, ¶44.

Between the publication of RFC 929 in 1984 and the October 14, 1996 critical date for the 1997 provisional application to which the 948 patent claims priority, a

great deal of work was published in the area of protocol offloading and optimization. *See* Ex.1003, ¶¶45-80. One common implementation, called Header Prediction, examined TCP/IP packet headers to determine whether a received packet was a normal one, meaning: the connection was in the ESTABLISHED state; no TCP control flags (SYN, FIN, RST, and URG) were set; and the packet was received in order. *Id.* ¶¶58-60. These normal packets were processed on the fast path. *Id.* ¶58. Header Prediction for fast path processing on a NIC specifically teaches the alleged ideas claimed in the 948 Patent.

TCP Header Prediction was developed by Van Jacobson, who wrote the implementation of Header Prediction in the freely distributed and widely used BSD TCP/IP stack available in 1996. *Id.* ¶59. Jacobson's Header Prediction is discussed in all of three of the references relied upon in this petition.

D. Direct Memory Access

Direct Memory Access (DMA) has been a well-known hardware technique that allows peripherals including network interface devices to directly access host memory without interrupting the host CPU. Ex.1003, ¶81. After received TCP/IP packets are processed on the fast path by separate hardware, the resulting data must be accessible to the application running on the host machine. *Id.* ¶82. DMA was a common and efficient way to achieve this. *Id.* Tanenbaum96 suggests the use of direct copying into the user buffer to avoid unnecessary copying. Ex.1006, .585.

The TCP/IP transport layer reassembles the original byte stream for use by the application layer after the headers are processed. *Id.* at .055-.056, .541; Ex.1003, ¶88.

5. OVERVIEW OF THE 948 PATENT

The 948 Patent relates generally to offloading TCP protocol processing for established TCP connections to a network interface card (NIC), which performs all protocol processing through the TCP layer and stores the packet payload data directly in its destination in application memory, bypassing the host protocol stack. The 948 Patent refers to the disclosed network interface card as an "intelligent network interface card (INIC)." Ex.1001, at Abstract. The INIC permits two modes of 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. This concept is illustrated in Fig. 6, shown and described below:

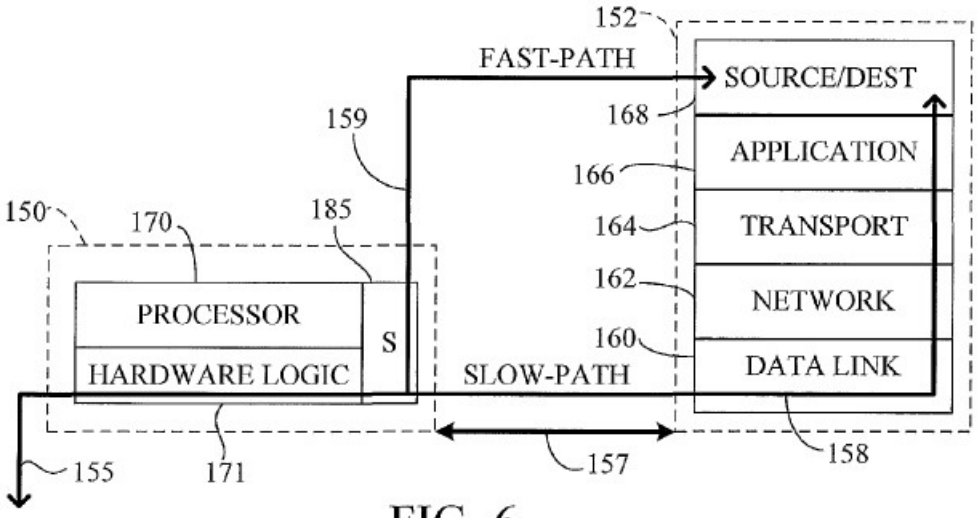


FIG. 6

A simplified intelligent network interface card (INIC) 150 is shown in FIG. 6 to provide a network interface for a host 152. Hardware logic 171 of the INIC 150 is connected to a network 155, with a peripheral bus (PCI) 157 connecting the INIC and host. The host 152 in this embodiment has a TCP/IP protocol stack, which provides a slow-path 158 for sequential software processing of message frames received from the network 155. The host 152 protocol stack includes a data link layer 160, network layer 162, a transport layer 164 and an application layer 166, which provides a source or destination 168 for the communication data in the host 152…. The INIC 150 has a network processor 170 which chooses between processing messages along a slow-path 158 that includes the protocol stack of the host, or along a fast-path 159 that bypasses the protocol stack of the host.

Ex.1001, Fig. 6; 9:11-29.

When a connection is created, the host computer creates a connection record which the 948 patent calls a "Communication Control Block (CCB)." *Id.* at 5:22-

29. Like the Transmission Control Block (TCB) described in the TCP Specification RFC 791, the CCB of the 948 Patent contains connection and state information for the connection. *Id.* at 12:32-39. When the INIC receives a packet, it looks for a CCB to determine if a connection exists, and if so, processes the packet on the fast-path, bypassing the host protocol stack:

> The processor 170 chooses, for each received message packet held in storage 185, whether that packet is a candidate for the fast-path 159 and, if so, checks to see whether a fast-path has already been set up for the connection that the packet belongs to. To do this, the processor 170 first checks the header status summary to determine whether the packet headers are of a protocol defined for fast-path candidates.… For fast-path 159 candidates, the processor 170 checks to see whether the header status summary matches a CCB held by the INIC. If so, the data from the packet is sent along fast-path 159 to the destination 168 in the host. If the fast-path 159 candidate's packet summary does not match a CCB held by the INIC, the packet may be sent to the host 152 for slow-path processing to create a CCB for the message.

Id. at 11:57-12:10.

The claims of the 948 Patent are directed to this notion of fast-path TCP receive processing when a connection is in the ESTABLISHED state.

6. 948 PATENT PROSECUTION HISTORY

The 948 Patent was filed on September 26, 2013 and issued less than 11 months later on August 12, 2014. There were no rejections or amendments.

The 948 Patent is a continuation of U.S. Patent Application No. 09/692,561, filed October 18, 2000, which is a continuation of U.S. Patent No. 6,226,680, filed April 28, 1998, which claims the benefit of U.S. Patent Application No. 60/061,809 filed October 14, 1997. Therefore, the earliest possible priority date of the 948 Patent is October 14, 1997.

On December 19, 2013, Applicant filed an Information Disclosure Statement with 383 patents, 41 applications, 13 foreign patents, and 112 non-patent literature documents, including Thia and Tanenbaum96. *See* Ex.1002, .075-.099. Neither Thia nor Tanenbaum96 were discussed during the prosecution of the application leading to the 948 Patent.

The Examiner's reasons for allowance follow:

> None of the prior art of record taken singularly or in combination teaches or suggest a network interface of a host computer for checking whether received packets have certain exception conditions, including whether the packets are IP fragmented, have FIN flag set, or out of order; processing any of the received packet that have the exception conditions, and storing payload data of the received packets that do not have any of the exception conditions in a buffer of the host computer and without any TCP header stored between the payload data of the received packets.

Ex.1002, .117.

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 948 Patent. 37 C.F.R. § 42.100(b).

7.2. Construction of Claim Terms

No relevant issues of claim construction are presented in the claims of the 948 Patent, and all terms should therefore simply be given their broadest reasonable construction in light of the specification as commonly understood by those of ordinary skill in the art.

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 948 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, ¶19.

9. DESCRIPTION OF THE PRIOR ART

9.1. Thia[2]

Thia teaches a hardware protocol engine for fast-path data transfer, bypassing the host protocol stack:

> Abstract - The Reduced Operation Protocol Engine (ROPE) presented here offloads critical functions of a multiple-layer protocol stack, based on the "bypass concept" of a fast path for data transfer.

Ex.1015, .001. Thia is based on the Open Systems Interconnect (OSI) layered networking model.

Thia describes a hardware protocol offload system that compares multiple incoming packet headers, including consecutive packets, with a template that identifies predicted bypassable headers using a receive bypass test (RX bypass test), and performs all protocol processing for packets in the "data transfer phase," bypassing the standard protocol stack (SPS) (host protocol stack):

> The receive bypass test matches the incoming PDU headers with a template that identifies the predicted bypassable headers. The bypass stack performs all the relevant protocol processing in the data transfer phase.

[2] Thia was published no later than March 20, 1996 and is prior art under 102(b). *See* Ex.1064.

Ex.1015, .003. The "receive bypass test" is performed on the Network Interface Adapter. Ex.1015, .006. The hardware is referred to as a reduced operation protocol engine ("ROPE") because it only operates on a subset of packets. Ex.1003, ¶100.

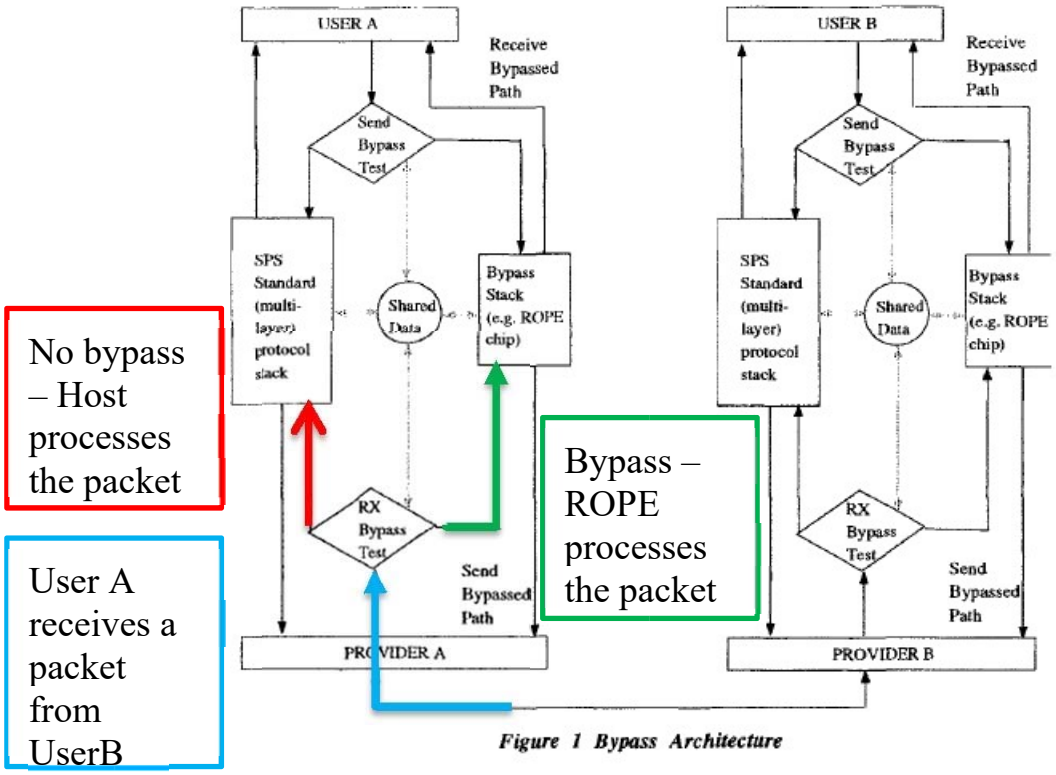


Figure 1 Bypass Architecture

Id. Fig. 1 (annotated).

Thia teaches that the receive bypass test ensures that the protocol bypass on the ROPE only handles normal packets − those in the data transfer phase (this is what Stevens2 referred to as the ESTABLISHED state for TCP) and that the SPS handles other phases:

> A multiple-layer bypass path is a concatenation of processing procedures performed by the adjacent layers when they are simultaneously in the data transfer phase. Meanwhile, the separate

> layers in the SPS [Standard Protocol Stack] path handle the other phases.

Ex.1015, .004.

In summary, the separation of the bypass path offers the following advantages:

> · The processing path of data PDUs can be optimized;
>
> · The number of possible PDU [Protocol Data Unit i.e. packet] formats in the bypass path is reduced to data transfer PDUs;
>
> · The finite state machine of the protocol is now reduced to only the "OPEN" state, for as long as processing remains in the bypass path. The state of the system does not change during the entire data transfer phase and the protocol processing is reduced to ensuring reliable transfer of data across the communications network.

Id. at .004. Thia's receive bypass test is a generalization of Jacobson's well-known "Header Prediction Algorithm" for TCP/IP, which is also described in the Tanenbaum96 and Stevens2 references discussed below. *Id.* at .002-.04.

The bypass stack on the ROPE performs header decoding and can perform the checksum. *Id*., .006.

Layer	*Procedure*	***Bypass Chip***	*Host*	*Per-Octet (A)*	*Per-Packet (B)*	*Per-Group-Of-Packets Aggregated to Per-Packet for bulk data transfer (B)*	*Remarks*
Presentation	Encoding	X		X			
	Encryption	X		X			
	Compression	X		X			
	Context Alteration		X			X	
Session	Synchronization Management		X			X	
	Token management		X			X	
Transport (Class 4)	Checksum (Optional)	X		X			
	Timer Management	X			X	X	Depends on Implementation
	Generation of ACK packets (Flow Control)	X				X	
	Resequencing	X			X		
All 3 layers	Header Construction	X			X		
	Header Decode	X			X		
	Buffer Management	X			X		Minimized (Simple scheme)
	Context Switching	X			X	X	Moved away from host OS
	Data Copying	With multiple-layer bypass, data Copying within layers is elminated.			X		Use of dual-ported memory and DMA..

Table 1 Bypassable versus Non-bypassable functions

Ex.1015, Table 1.

Thia's DMA copies the processed data from the bypass stack to Host Memory.

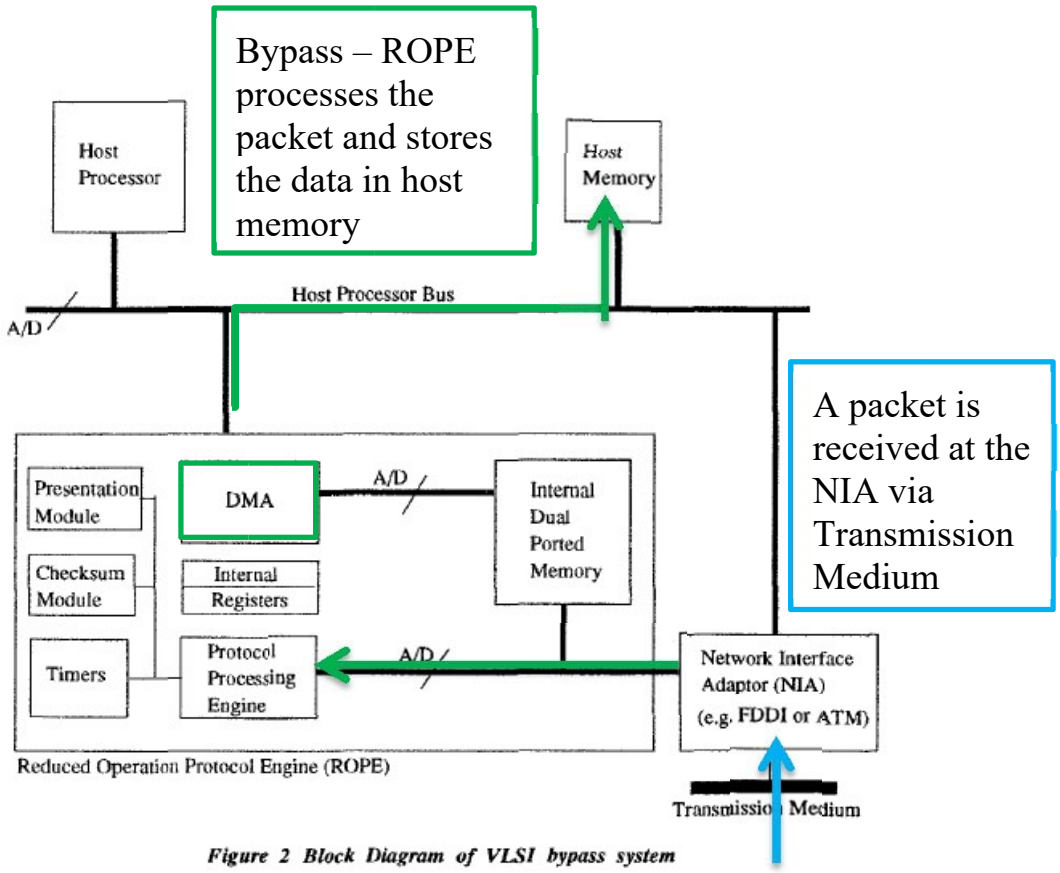


Figure 2 Block Diagram of VLSI bypass system

Id. at Fig. 2 (annotated).

As show in Table 1 above, data copying within layers is eliminated.

9.2. Tanenbaum96: A. Tanenbaum, Computer Networks, 3rd ed. (1996)[3]

As described by the 948 Patent, Tanenbaum96 is a textbook devoted primarily to layered protocol processing. Ex.1001, 3:4-19. It is a widely-cited textbook

[3] Tanenbaum96 was published no later than August 9, 1996 and is prior art under 35 U.S.C. § 102(b). Ex.1006; *see also* Ex.1011. Tanenbaum96 was provided during prosecution in an IDS listing 383 patents, 41 applications, 13 foreign patents, and

covering network hardware, software, protocols (including OSI and TCP/IP) and standards. Ex.1003, ¶105. It was published no later than August 9, 1996 and is therefore prior art to the October 14, 1997 provisional application to which the 948 Patent claims priority. *See* Ex.1011.

Tanenbaum96 recognizes that an "obstacle to fast networking is protocol software," and teaches "fast path" processing for TCP as a solution for normal packets. Ex.1006, .583-84; Ex.1003, ¶107. Tanenbaum96 teaches TCP fast path receive processing by looking up a TCP connection record, checking to see if it, the packet, is a normal one in the ESTABLISHED state, that the data is not fragmented, no special flags are set, and the packet is not out of sequence, and then putting the data directly into user memory. This well-known scheme was called "header prediction":

> 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

---

112 non-patent literature documents. *See* Ex.1002 at .092. It was not relied on by the Examiner during prosecution of the 948 Patent.

> 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,[4] 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. … 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, .584-.585 (underlining added, bold in original). The "connection record" is used to maintain TCP state for each connection. *Id.* at .549. A structure for maintaining TCP state information is also called a "Transmission Control Block (TCB)" as described in RFC 793, the TCP Protocol Specification. Ex.1007, .024.

As the quotation above notes, header prediction was included in many TCP implementations in 1996. This included the widely used BSD implementation described in Stevens2 below. The original developer of header prediction, Van

[4] As discussed below, the RST and FIN flags close the connection, i.e. take it out of the ESTABLISHED state. The SYN flag establishes a connection (i.e., the ESTABLISHED state), meaning that a SYN packet is not a part of a connection in the ESTABLISHED state. Ex.1006, .584-585.

Jacobson, implemented header prediction in the BSD TCP/IP protocol stack. Ex.1013, .960 ("*Header Prediction* was put into the 4.3BSD Reno release by Van Jacobson"), .961-.962.[5]

Tanenbaum96 shows the finite state machine used by TCP/IP to maintain a connection. The arrows indicate how the state changes in response to the flags (SYN, FIN, ACK, RST) in the TCP header:

[5] The 948 Patent claims priority to a 1997 Provisional, which indicates that "header prediction" in FreeBSD was the basis for the receive fast-path processing: "The base for the receive processing done by the INIC on an existing context is the fast-path or "header prediction" code in the FreeBSD release." Ex.1031, .057.

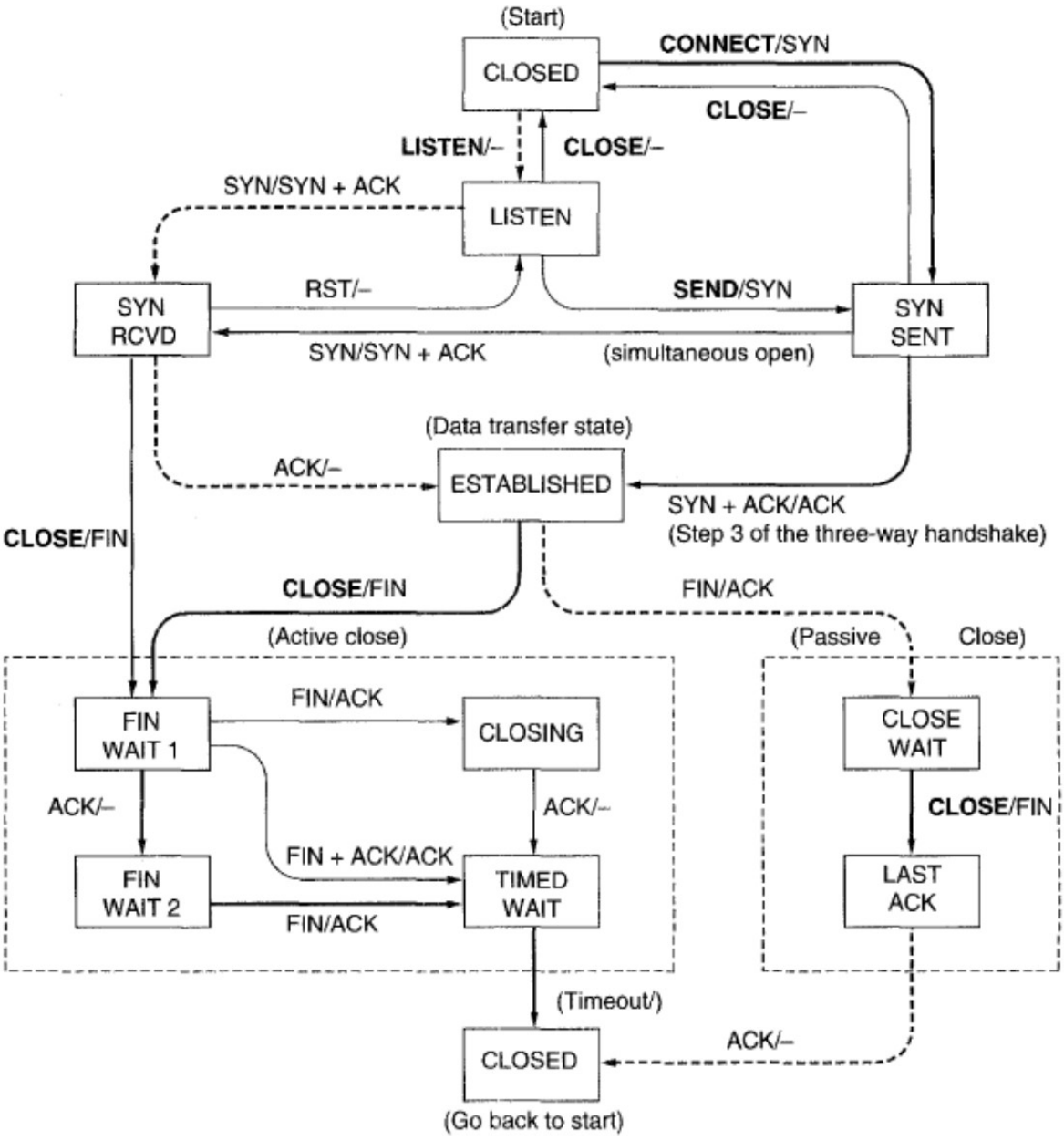


Fig. 6-28. TCP connection management finite state machine. The heavy solid line is the normal path for a client. The heavy dashed line is the normal path for a server. The light lines are unusual events.

Ex.1006, Fig. 6-28

The SYN, FIN and RST flags directly affect whether a TCP connection is in the ESTABLISHED state. SYN (synchronize) establishes a connection, FIN

(finished) releases a connection, and RST (reset) is "used to reset a connection that has become confused due to a host crash or some other reason." Ex.1006, .545, Fig. 6-24. These flags in the TCP header are illustrated below in Fig. 6-24 of Tanenbaum96 (annotated).

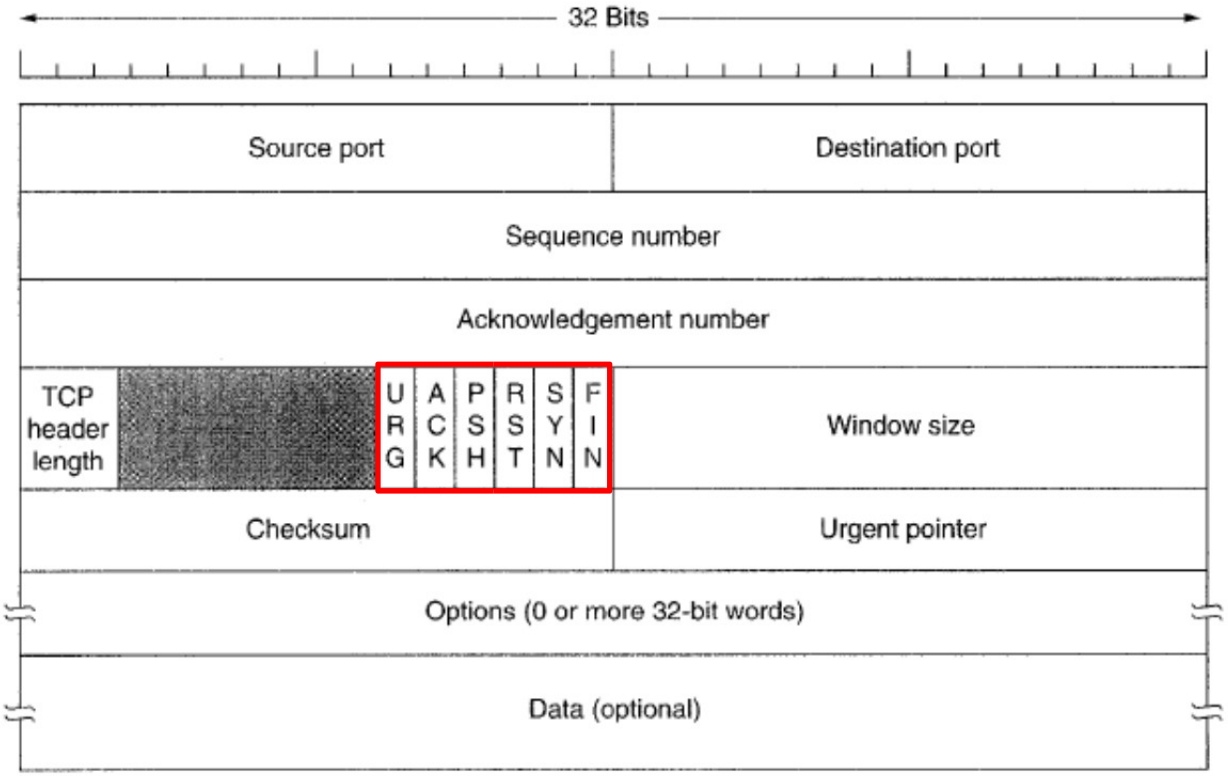


Fig. 6-24. The TCP header.

Id. Fig. 6-24

Tanenbaum96 also discloses the benefits of limiting the amount of unnecessary data copying, including between multiple layers before it reaches the receiving application process. *Id.* at .579, .582; *Id.* at .590 ("As we saw earlier,

copying data is often the main source of overhead. Ideally, the hardware should dump each incoming packet into memory as a contiguous block of data.").

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).

9.3. Stevens2: TCP-IP Illustrated, Vol.2[6]

Stevens2 is one of the most widely-read and referenced books on the implementation of TCP/IP. Ex.1003, ¶123. It guides the reader through the source code for the implementation of TCP/IP in the 4.4BSD-Lite distribution, which was widely-used to implement TCP/IP by companies designing networking products. *Id.* The discussion covers BSD's implementation of Jacobson's Header Prediction, which was written by Jacobson himself. Ex.1013, .960 ("*Header Prediction* was put into the 4.3BSD Reno release by Van Jacobson"), .961-.962 ("Header prediction helps unidirectional data transfer by handling the two common cases. . . If TCP is

[6] Stevens2 was published no later than April 7, 1995 and is prior art under 102(b). *See also* Ex.1063.

receiving data, the next expected segment for this connection is the next in-sequence data segment. . . a small set of tests determines if the next expected segment has been received, and if so, it is handled in-line, faster than the general processing that follows.")

tcp_input.c

```
347     /*
348      * Header prediction: check for the two common cases
349      * of a uni-directional data xfer.  If the packet has
350      * no control flags, is in-sequence, the window didn't
351      * change and we're not retransmitting, it's a
352      * candidate.  If the length is zero and the ack moved
353      * forward, we're the sender side of the xfer.  Just
354      * free the data acked & wake any higher-level process
355      * that was blocked waiting for space.  If the length
356      * is non-zero and the ack didn't move, we're the
357      * receiver side.  If we're getting packets in order
358      * (the reassembly queue is empty), add the data to
359      * the socket buffer and note that we need a delayed ack.
360      */
361     if (tp->t_state == TCPS_ESTABLISHED &&
362         (tiflags & (TH_SYN | TH_FIN | TH_RST | TH_URG | TH_ACK)) == TH_ACK &&
363             (!ts_present || TSTMP_GEQ(ts_val, tp->ts_recent)) &&
364             ti->ti_seq == tp->rcv_nxt &&
365             tiwin && tiwin == tp->snd_wnd &&
366             tp->snd_nxt == tp->snd_max) {
367             /*
368              * If last ACK falls within this segment's sequence numbers,
369              *  record the timestamp.
370              */
371             if (ts_present && SEQ_LEQ(ti->ti_seq, tp->last_ack_sent) &&
372                 SEQ_LT(tp->last_ack_sent, ti->ti_seq + ti->ti_len)) {
373                 tp->ts_recent_age = tcp_now;
374                 tp->ts_recent = ts_val;
375             }
```

tcp_input.c

Figure 28.11 tcp_input function: header prediction, first part.

The following six conditions must *all* be true for the segment to be the next expected data segment or the next expected ACK:

1. The connection must be ESTABLISHED

2. The following four control flags must not be on: SYN, FIN, RST, or URG.

***

4. The starting sequence number of the segment (ti_seq) must equal

> the next expected receive sequence number (rcv_nxt). If this test is false, then the received segment is …a segment beyond the one expected.

Id. at .962-963 (emphasis added). Header Prediction checks to see if a packet is eligible for fast path processing by ensuring that a connection is established; the SYN, FIN, and RST flags are not set; and the sequence number is in order (i.e., it is the next packet in a data transfer that does not need special processing). The code above is implemented at the TCP layer, and thus, does not check for IP fragmentation because the IP layer reassembles fragmented IP packets. Ex.1003, ¶126.

9.4. Motivations To Combine

9.4.1. Thia in combination with Tanenbaum96

Thia discloses a chip for fast path receive protocol processing and a bypass test to offload protocol processing of packets in the data transfer phase for the OSI Session and Transport layer protocols. Ex.1015, .001, .003. Thia discloses that its protocol stack bypass could be used with "any standard protocol." *Id.* at .003. A POSA would be motivated to use Thia's fast path protocol processing for TCP/IP, which was among the most popular transport protocols in the world in 1996, to achieve the many benefits described by Thia, including eliminating "inter-layer operations such as queue and buffer management, context switching, and the movement of data across layers, all of which are a significant overhead." *Id.*at .001.

A POSA seeking to use Thia's chip with TCP/IP would naturally look to a well-known "simplified . . . college-level textbook devoted primarily to th[e] subject . . ., such as Computer Networks, Third Edition (1996) by Andrew S. Tanenbaum." Ex.1001, 3:4-8; Ex.1003, ¶128. Tanenbaum96 addresses both the OSI and TCP/IP models, and notes that "the TCP/IP internet layer is very similar in functionality to the OSI network layer," and that the TCP/IP transport layer "is designed to allow peer entities on the source and destination hosts to carry on a conversation, the same as in the OSI transport layer." Ex.1006, .054. Tanenbaum96 also discloses that TCP/IP became the dominant protocol suite as the OSI model vanished. *Id.* at .016. A POSA would thus be motivated to use Thia for TCP/IP. Ex.1003 at ¶131.

Importantly, Thia discloses that its fast path protocol bypass is a generalization of Jacobson's Header Prediction Algorithm for TCP/IP. *Id.* at .002. Thia states that its "receive bypass test" matches incoming headers with a template that identifies the predicted bypassable headers. *Id.* at ¶130.

Tanenbaum96 discloses Header Prediction and states that "[m]any TCP implementations use it." Ex.1006, .584-.585. Tanenbaum96 describes that Header Prediction determines if the received packet is a normal one and describes the conditions to check make that determination. Packets meeting those conditions are eligible for fast path processing. This includes verifying that there is a connection record matching the two IP addresses (one for source and one for destination) and

two ports (one for source and one for destination) found in the header of the packet. *Id.* at .585. Additionally, the connection "state is *ESTABLISHED*, neither side is trying to close the connection, TPDU is a full one [i.e., not fragmented], no special flags are set, and the sequence number is the one expected." *Id*.

A POSA reading Thia in combination with Tanenbaum96 would have been motivated to use Tanenbaum96's conditions to determine normal packets eligible for fast path processing as part of Thia's received bypass test because, as clarified in Section 9.4.2, both are based on Jacobson's TCP/IP Header Prediction that is used to improve performance by bypassing standard protocol processing for normal data transfers that do not need special processing and avoiding multiple data copies and encoding associated with multilayer processing. Ex.1003, ¶132; *see* Ex.1006, .584-.585; Ex.1015, .002. Further, a POSA would have been motivated to reduce the burden of protocol processing on the host processor such that it may perform other necessary functions. Thus, a POSA would have been motivated to migrate to TCP/IP and to offload processing for normal data transfers from a host processor to separate hardware and implement direct memory access by storing data (stripped of the TCP headers) to eliminate interlayer copying of data and improve performance. Ex.1003, ¶132.

9.4.2. Thia in combination with Tanenbaum96 and Stevens2

A POSA would have been motivated to combine Stevens2 with Thia, in combination with Tanenbaum96. Tanenbaum96 expressly references the sibling volume of Stevens2 as providing a comprehensive treatment of TCP, IP and related protocols. Ex.1006, .790. Similarly, Stevens2 expressly references the 1989 second edition of Tanenbaum96 in its bibliography. Further, like Tanenbaum96, Stevens2 is another well-known textbook concerning the layered protocol TCP/IP. Stevens2 reproduces and explains Jacobson's implementation of Header Prediction in freely available and widely used BSD TCP/IP protocol stack implementation. Ex.1013, .960-.963; Ex.1003, ¶133.

Stevens2 discloses that Header Prediction improves unidirectional data transfer by handling the common (normal) case where TCP is receiving data and the next expected segment for the connection is the next in-sequence data segment. Ex.1013, .962. Header Prediction confirms the next expected segment has been received by checking whether the connection is ESTABLISHED; the SYN, FIN, and RST flags are not set, and the segment is not out of order, consistent with the description of fast path processing in Tanenbaum96. *Id.* at .962-.963.

A POSA would be motivated to combine Thia's receive bypass test with the conditions of Jacobson's Header Prediction as explained in Tanenbaum96 and Stevens2 because a POSA would want to reduce the burden of protocol processing

on the host processor to reduce the bottleneck recognized by Thia and Tanenbaum96 such that it may perform other necessary functions. Ex.1003, ¶133. Thus, a POSA would be motivated to use Thia's separate hardware to offload processing from a host processor to efficiently process normal/expected packets that have a pre-established connection in the data transfer mode. *Id*. at 135.

10. GROUND #1:

As set forth below, Thia in combination with Tanenbaum96 in further combination with Stevens2 renders obvious claims 1, 3, 6-9, 11, 14-17, 19, and 21-22 of the 948 Patent. Thia discloses receiving and checking transport layer packets on a NIC to determine if they are in the data transfer phase, using the Reduced Operation Protocol Engine ("ROPE") chip (fast path that bypasses multiple protocol layer processing on the host) to perform protocol processing for packets in the data transfer phase and the Standard Protocol Stack (SPS, slow path) for others, and using DMA to move data from the ROPE chip to the host memory. Ex.1003, ¶¶ 127-132.

Tanenbaum96 discloses a TCP/IP protocol stack, the details of a TCP/IP connection and headers, and using Header Prediction to perform fast path protocol processing for packets in the ESTABLISHED state (i.e. data transfer phase) and copying received data to the user to avoid multiple data copies between layers. *Id.* ¶¶129-130. Stevens2 provides additional details of Jacobson's Header Prediction, which is referenced as the basis for the fast path processing in all three references.

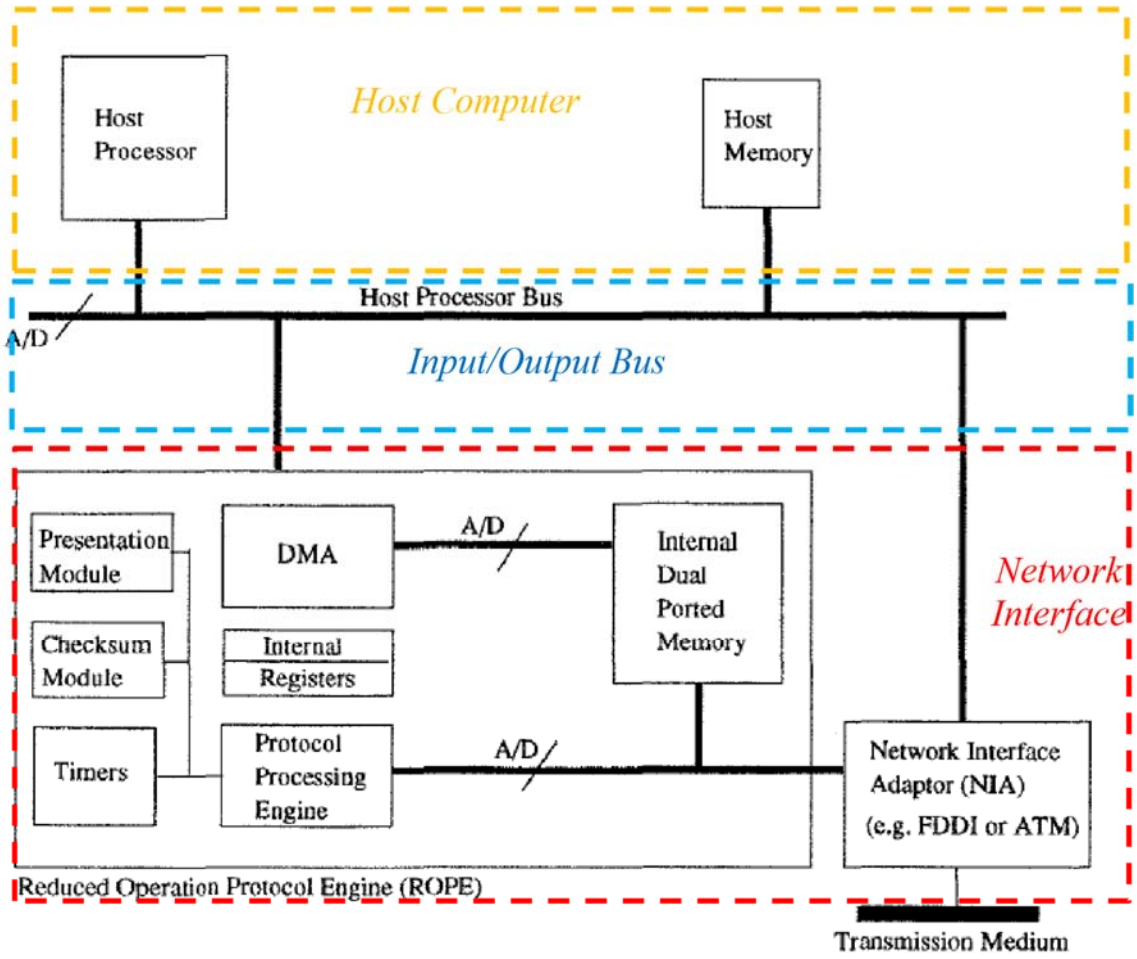
Id. ¶¶133-135. In combination with Tanenbaum96 and Stevens2, it was obvious to implement Thia's concept of hardware bypass to conditionally bypass host processing of TCP/IP packets in the ESTABLISHED state using Header Prediction and subsequently move the payload data into host memory via DMA, thereby rendering obvious claims 1, 3, 6-9, 11, 14-17, 19, and 21-22. *Id.* ¶¶127-135.

10.1. Claim 1

10.1.1. [1.P] A method for network communication by a host computer having a network interface that is connected to the host by an input/output bus, the method comprising

To the extent that the preamble is limiting, Thia in combination with Tanenbaum96 in further combination with Stevens2 discloses *a method for network communication by a host computer having a network interface that is connected to the host by an input/output bus.*

Specifically, Thia discloses a Reduced Operation Protocol Engine ("ROPE") chip and a Network Interface Adapter ("NIA") (together "*a network interface*"), both of which are connected to the Host Processor and Host Memory (together the "*host computer*") by a Host Processor Bus ("*an input/output bus*"), as shown below:



Ex.1015, Fig. 2 (annotated).

A POSA would have understood that the combination of the ROPE chip with the NIA is the "*network interface*" because they operate together and provide the interface between the Transmission Medium (used for "*network communication*") and the Host Processor and Host Memory.

10.1.2. [1.1] running, on the host computer, a protocol processing stack including an Internet Protocol (IP) layer and a Transmission Control Protocol (TCP) layer, with an application layer running above the TCP layer;

Thia in combination with Tanenbaum96 in further combination with Stevens2 discloses this limitation.

10.1.2.1. running, on the host computer

As explained above in Section 9.1, Thia discloses a Standard Protocol Stack ("SPS") *running on the host computer*. Specifically, Thia discloses a bypass architecture in which a "receive bypass test" (RX Bypass Test) on the Network Interface Adapter (NIA) (Ex.1015, .006), determines whether a received packet is to be processed by a fast path bypass stack on the ROPE chip or the SPS on the host.

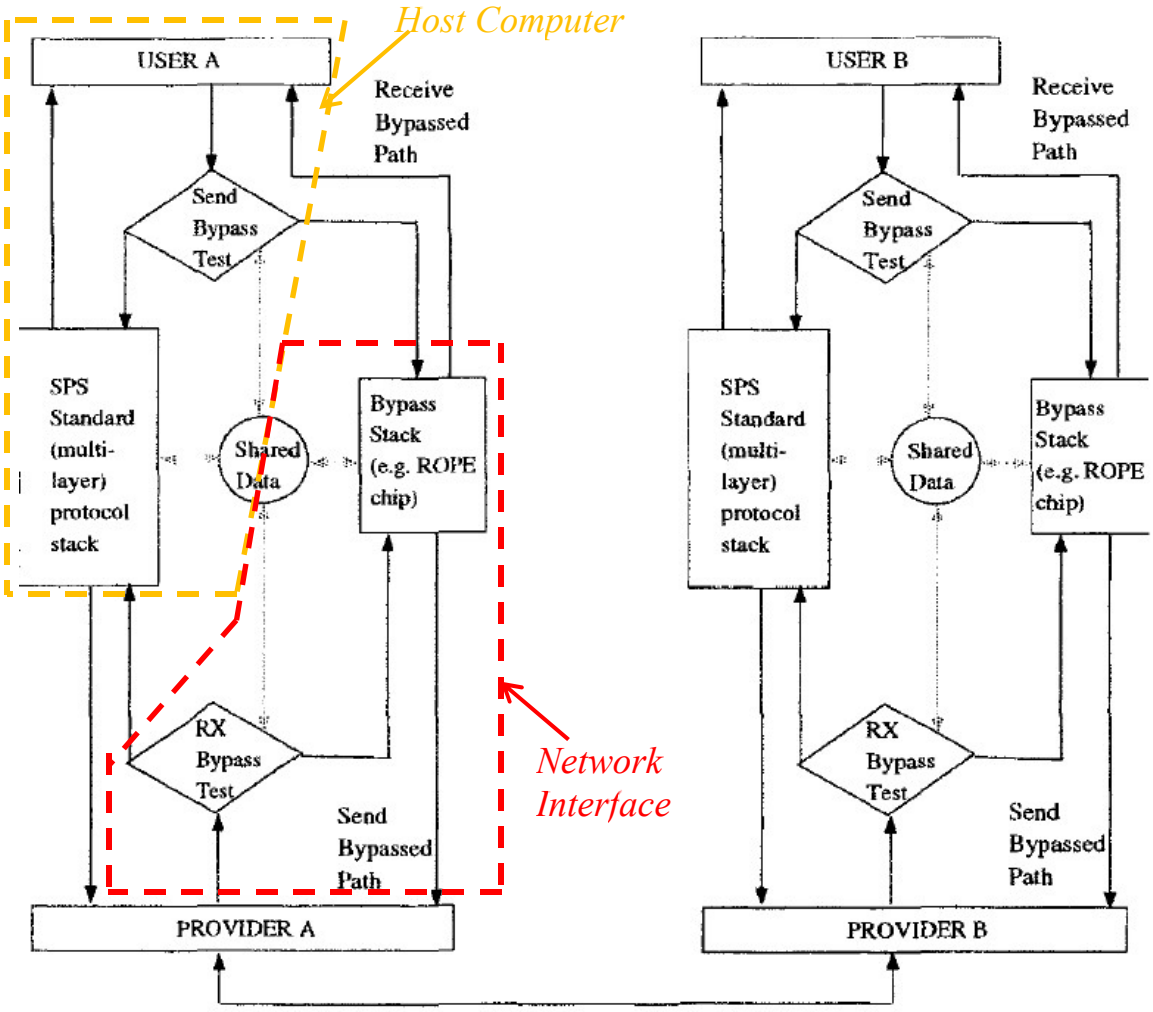


Figure 1 Bypass Architecture

Id. Fig. 1 (annotated).

See also id. at .010 ("Whenever the **host processor encounters a switch** in the processing path, i.e., from the bypass stack **to the SPS**. . ." (emphasis added), .002 ("The contribution of this paper is to define the host/chip interface and the chip operation.").

10.1.2.2. a protocol processing stack including

Thia discloses that the SPS protocol processes packets that are not bypassed. Ex.1015, .003 ("The standard protocol stack (SPS) is the processing path taken by all PDU's during a connection without the bypass.").

10.1.2.3. an Internet Protocol (IP) layer and a Transmission Control Protocol (TCP) layer

Thia in combination with Tanenbaum96 discloses *an Internet Protocol (IP) layer and a Transmission Control Protocol (TCP) layer.* Thia discloses a hardware offload of a multiple-layer protocol stack (the SPS), and specifically discloses the design of a ROPE chip for the OSI Session and Transport layer protocols. Ex.1015, .001. As explained above in Section 9.4.1, a POSA working with the protocol processing of layered protocols in the OSI model would look to Tanenbaum96 and have been motivated to migrate to the more dominant TCP/IP protocol suite as OSI models vanished. Ex.1003, ¶128. Thus, the SPS would include IP and TCP layers in place of the OSI Network and Transport layers and the Session layer and Presentation layer could be incorporated into the Application layer. *Id*; *see* Ex.1006, Tanenbaum96 at .054 ("the TCP/IP internet layer is very similar in functionality to the OSI network layer . . . [t]he layer above the internet layer in the TCP/IP model is now usually called the transport layer. It is designed to allow peer entities . . . to carry on a conversation, the same as in the OSI transport layer"), .055 ("[the session

and presentation layers] are of little use to most applications."). *Id.* at .054. Thia's bypass test is based on Jacobson's TCP/IP Prediction Header.

10.1.2.4. with an application layer running above the TCP layer;

As illustrated above in the Background section, the OSI and TCP/IP models have application layers above the transport layer.

Ex.1003, ¶24.

Accordingly, Thia in combination with Tanenbaum96 in further combination with Stevens2 discloses a host processor and host memory (together *a host computer*) with a Standard Protocol Stack (*running a protocol processing stack*) *including Internet Protocol and Transmission Control Protocol (TCP) layers with an Application Layer running above the TCP layer*. *Id.* ¶128.

10.1.3. [1.2] initializing, by the host computer, a TCP connection that is defined by source and destination IP addresses and source and destination TCP ports;

As shown by the following sections, Thia in combination with Tanenbaum96 in further combination with Stevens2 discloses this limitation.

10.1.3.1. initializing, by the host computer a TCP connection

Thia discloses initializing connections by a host. Ex.1015, .004. As explained in Section 10.1.2.3, a POSA would have been motivated to implement Thia's SPS using the TCP/IP protocol suite.

As explained in Section 4.B, a TCP/IP connection is initialized between two machines as illustrated in Fig. 6-28.

Ex.1006, Fig. 6-28.

The SYN flag is used to initialize a connection, which transitions into the ESTABLISHED state. *Id.* at .545, .550. A packet with a SYN flag fails the conditions of the receive bypass test because it is not in the ESTABLISHED state and processing must be performed by the SPS on the host. Accordingly, the host is responsible for initializing the TCP connection. *Id.*, .567; Ex.1003, ¶36.

10.1.3.2. that is defined by source and destination IP addresses and source and destination TCP ports

As explained in Section 4.B, a TCP/IP connection is defined by the IP address and TCP port for both sides of the connection. *See also* Ex.1006, .541 ("TCP service is obtained by having both the sender and receiver create end points, called sockets . . . [e]ach socket has a socket number (address) consisting of the IP address of the host and a 16-bit number local to that host, called a port.")

Accordingly, Thia in combination with Tanenbaum96 in further combination with Stevens2 discloses initializing, by the host computer (SPS), a TCP connection that is defined by source and destination IP addresses and source and destination TCP ports.

10.1.4. [1.3] receiving, by the network interface, first and second packets, wherein the first packet has a first TCP header and contains first payload data for the application, and the second packet has a second TCP header and contains second payload data for the application;

As shown by the following sections, Thia in combination with Tanenbaum96 in further combination with Stevens2 discloses this limitation.

10.1.4.1. receiving, by the network interface, first and second packets

As explained above in Section 10.1.1, Thia's Network Interface Adapter, together with the ROPE chip, is the *network interface*. As explained above in Section 9.1, the NIA receives multiple packets, including consecutive packets:

> The receive bypass test matches the incoming PDU headers with a template that identifies the predicted bypassable headers. The bypass stack performs all the relevant protocol processing in the data transfer phase.

Ex.1015, .003.

10.1.4.2. wherein the first/second packet has a first TCP header and contains first/second payload data

A network interface connected to a TCP/IP network (as explained above in Section 10.1.3.1) receives TCP/IP packets (segments), which contain TCP headers and optionally payload data. Tanenbaum96 illustrates this well-known structure. *See e.g.,* Ex.1006, .544-547.

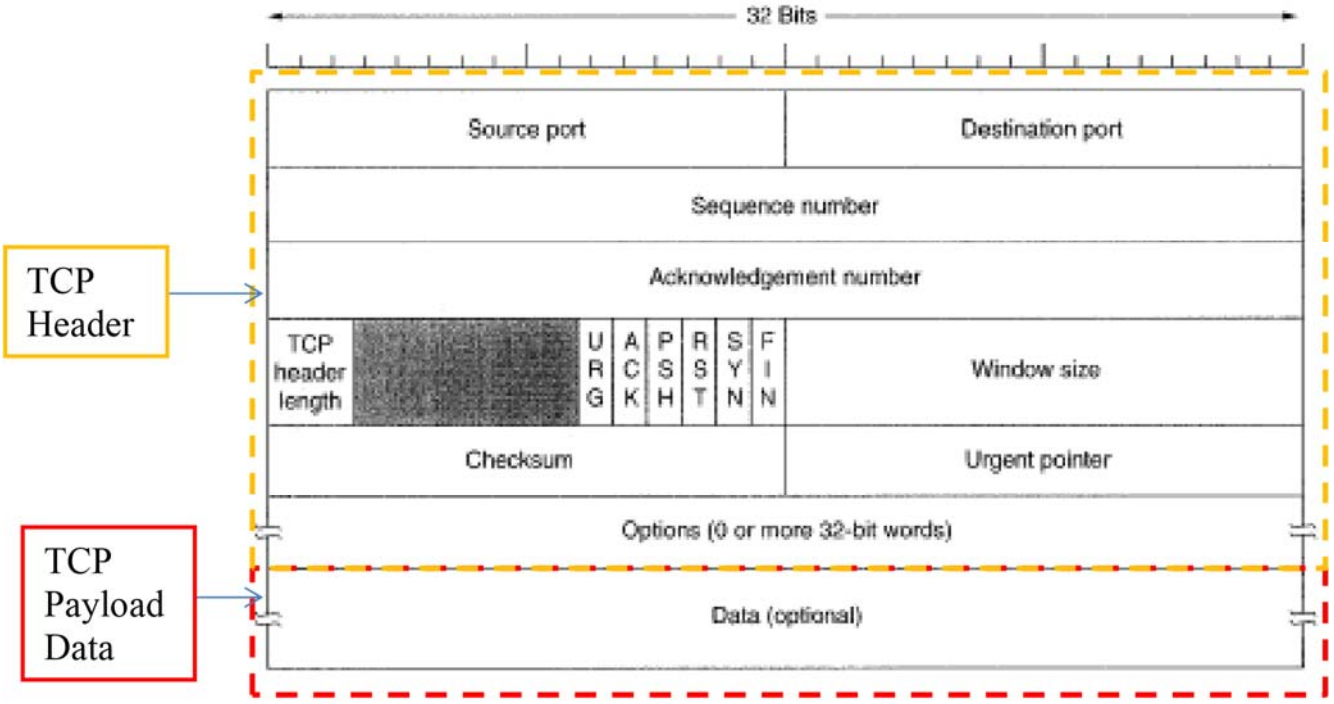


Fig. 6-24. The TCP header.

Id. Fig. 6-24 (annotated).

10.1.4.3. for the application

Thia discloses that the received data is for applications. *See* Ex.1015, .005 (protocol processing that might limit the "effective throughput [of data] **presented to the application processes**, especially for bulk data transfer.") (emphasis added); *see also id.* Fig. 1 (providing data to the user).

As explained above in Section 10.1.2.4, the application layer is above the Transport layer in TCP/IP and OSI model implementations. Ex.1006, .052-.053. Accordingly, modifying Thia to use the TCP/IP packets, the payload data in the TCP packet is for the Application layer (i.e., *for the application*). Ex.1003, ¶128.

Thus, Thia in combination with Tanenbaum96 in further combination with Stevens2 discloses Thia's NIA (part of the *network interface*) receiving packets (*receiving . . . first and second packets*, wherein the packets *have a TCP header and payload data for the application*).

10.1.5. [1.4] checking, by the network interface, whether the packets have certain exception conditions, including checking whether the packets are IP fragmented, checking whether the packets have a FIN flag set, and checking whether the packets are out of order;

As shown by the following sections, Thia in combination with Tanenbaum96 in further combination with Stevens2 discloses this limitation.

10.1.5.1. checking, by the network interface

As explained above in Section 9.1, Thia's NIA (part of the *network interface*) performs the receive bypass test (*checking*).

10.1.5.2. whether the packets have certain exception conditions, including checking whether the packets are IP fragmented, checking whether the packets have a FIN flag set, and checking whether the packets are out of order

As explained above in Sections 9.4.1 and 9.4.2, a POSA would have been motivated to implement Header Prediction as disclosed in Tanenbaum96 and Stevens2 as a part of Thia's receive bypass test. Thia's receive bypass test is a generalization of Jacobson's "Header Prediction Algorithm" for TCP/IP, which is also described in the Tanenbaum96 and Stevens2 references discussed below.

Ex.1015, .002. Header Prediction checks the TCP headers of received packets to determine if the packet is a normal one: the state is ESTABLISHED (i.e., *no SYN* flag), neither side is trying to close the connection (i.e., *no FIN or RST flag*), the TPDU is a full one (e.g*., no IP fragmentation*), no special flags are set (including the SYN, FIN and RST flags), and the sequence number is the one expected (i.e., *the packets are not out of order*). Ex.1003, ¶114; *see* Ex.1006, .583-.585 (many TCP implementations use it); 585 (disclosing the test above), .545 (description of flags); *see also* Ex.1013, .962-.963 (walkthrough of the BSD code for the test above, which tests whether control flags SYN, FIN, RST are set). Thia discloses that bypass processing is for simple bulk data transfer and excludes reassembly of fragmented packets, which should be restricted to the lower layers. Ex.1015, .002, .014.

Thus, Thia in combination with Tanenbaum96 in further combination with Stevens2 discloses the NIA (part of the *network interface*) performing a receive bypass test (*checking . . . whether the packets have certain exception conditions*) *including checking for whether the packets are IP fragmented, checking whether the packets have a FIN flag set, and checking whether the packets are out of order*.

10.1.6. [1.5] if the first packet has any of the exception conditions, then protocol processing the first TCP header by the protocol processing stack;

Thia in combination with Tanenbaum96 in further combination with Stevens2 discloses that if the first packet has any of the exception conditions, then protocol processing the first TCP header by the protocol processing stack.

As explained in Section 9.1, Thia teaches that packets that fail the receive bypass test are processed by the Standard Protocol Stack (SPS) (*the protocol processing stack*).

Ex.1015, Fig. 1.

Thus, packets failing the conditions of Header Prediction would fail the receive bypass test and be processed by the SPS.

10.1.7. [1.6] if the second packet has any of the exception conditions, then protocol processing the second TCP header by the protocol processing stack;

Thia in combination with Tanenbaum96 in further combination with Stevens2 discloses this limitation.

The same receive bypass test applies to all packets. Ex.1015, .003 ("The receive bypass test matches the incoming PDU headers with a template that identifies the predicted bypassable headers."). *See* Section 10.1.6 above.

10.1.8. [1.7] if the packets do not have any of the exception conditions, then bypassing host protocol processing of the TCP headers and storing the first payload data and the second payload data together in a buffer of the host computer, such that the payload data is stored in the buffer in order and without any TCP header stored between the first payload data and the second payload data.

As shown by the following sections, Thia in combination with Tanenbaum96 in further combination with Stevens2 discloses this limitation.

10.1.8.1. if the packets do not have any of the exception conditions, then bypassing host protocol processing of the TCP headers

As explained in Section 9.1, Thia teaches that packets that pass the receive bypass test are processed by the ROPE chip (part of the *network interface*), bypassing the SPS (on the *host computer*), for processing.

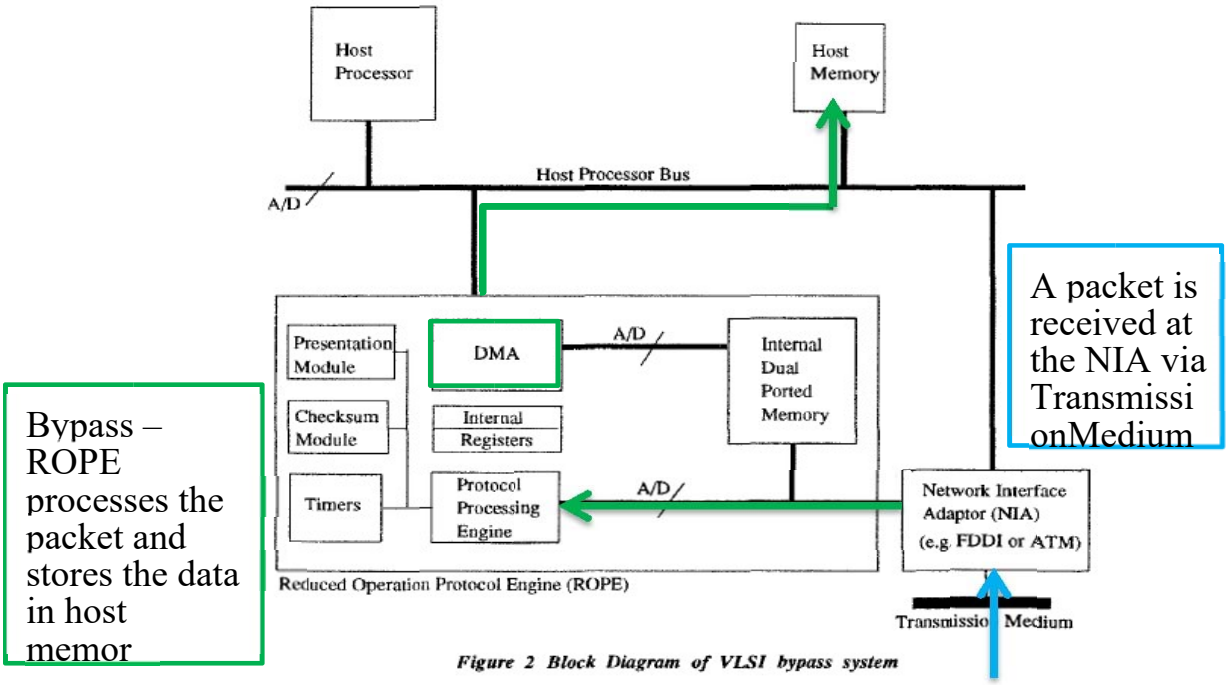
Ex.1015, Fig. 1.

10.1.8.2. and storing the first payload data and the second payload data together in a buffer of the host computer

Thia teaches the use of DMA to move packet data from the ROPE chip to host memory:

> Movement of data across the host bus interface are minimized by using an on-chip DMA for fast block data transfer to/from the host system memory.

Id. at .007.



Id. Fig. 2 (annotated).

Thus, as packets are processed by the ROPE chip, their payload data is moved by DMA from the ROPE chip to host memory (a *host buffer*). Ex.1003, ¶103.

10.1.8.3. such that the payload data is stored in the buffer in order and without any TCP header stored between the first payload data and the second payload data

Thia in combination with Tanenbaum96 in further combination with Stevens2 would be understood by a POSA to disclose storing payload data in order without intervening headers. Thia discloses that the ROPE chip (part of the *network interface*) is responsible for decoding the packet header and optionally the checksum and data copying within layers is eliminated.

Layer	*Procedure*	***Bypass Chip***	*Host*	*Per-Octet (A)*	*Per-Packet (B)*	*Per-Group-Of-Packets Aggregated to Per-Packet for bulk data transfer (B)*	*Remarks*
Presentation	Encoding	X		X			
	Encryption	X		X			
	Compression	X		X			
	Context Alteration		X			X	
Session	Synchronization Management		X			X	
	Token management		X			X	
Transport (Class 4)	Checksum (Optional)	X		X			
	Timer Management	X			X	X	Depends on Implementation
	Generation of ACK packets (Flow Control)	X				X	
	Resequencing	X			X		
All 3 layers	Header Construction	X			X		
	Header Decode	X			X		
	Buffer Management	X			X		Minimized (Simple scheme)
	Context Switching	X			X	X	Moved away from host OS
	Data Copying	With multiple-layer bypass, data Copying within layers is eliminated.			X		Use of dual-ported memory and DMA..

Table 1 Bypassable versus Non-bypassable functions

Ex.1015, Table 1 (annotated).

Thus, the header is already decoded and checked when the ROPE chip moves the data in a packet to Host Memory. *See* Ex.1003, ¶128. Thia discloses moving the data to host memory. Ex.1015, .007. A POSA would have appreciated that there is no need to transfer the already decoded and checked headers to host memory. *Id.* at .006; Ex.1003, ¶128.

Further, as taught by Tanenbaum, by the time the data reaches the application in host memory, it is stored as a byte stream. Ex.1003, ¶112. Tanenbaum96 discloses that an outbound byte stream of data is broken up and sent as separate IP datagrams (each with their own header). On receipt, the original byte stream is reconstructed (removing the headers from each datagram and arranging them in order) by a TCP entity on a network interface card (such as Thia's ROPE chip and NIA). *See* Ex.1006, .498 ("The transport entity can be . . . on the network interface card"), .540 ("A TCP entity accepts user data streams from local processes, breaks them up into pieces . . . and sends each piece as a separate IP datagram. When IP datagrams containing TCP data arrive at a machine, they are given to the TCP entity, which reconstructs the original byte streams.").

Also, the fast path disclosed in Tanenbaum96, consistent with the option in Thia, offloads the calculation of the checksum for the header, eliminating an extra pass over the data by higher layers. *Id.* at .585-.589 ("The header and data should be separately checksummed, for two reasons. First, to make it possible to checksum the header but not the data. Second, to verify that the header is correct before starting to copy the data into user space. It is desirable to do the data checksum at the time the data are copied to user space, but if the header is incorrect, the copy may be to the wrong process."). Therefore, there is no need to transfer the header data to the application. Ex.1003, ¶128.

Thus, a POSA would have been motivated to use Thia's protocol engine for a TCP *network interface* using the TCP disclosures found in Tanenbaum96 and Stevens2. *Id*. As explained above, when incoming packets pass the receive bypass test (*packets do not have any of the exception conditions*), they are processed by the ROPE chip as opposed to the SPS (*bypassing host protocol processing of the TCP headers*). The output from the ROPE chip is the original byte stream (*in order and without any TCP header stored between*). A POSA would utilize Thia's on-chip DMA to transfer only the data blocks together in a buffer in host memory (*storing . . . together in a buffer of the host computer*).

10.2. Claim 3

10.2.1. [3] The method of claim 1, wherein storing the first payload data and the second payload data together in a buffer of the host computer is performed by a direct memory access (DMA) unit of the network interface.

Thia in combination with Tanenbaum96 in further combination with Stevens2 discloses the method of claim 1 (*See* Section 10.1) and this limitation.

As explained in Section 10.1.8.2, Thia teaches the use of DMA to move packet data from the ROPE chip to host memory (a *host buffer*). Thia also teaches that "data Copying within layers is eliminated." Ex.1015, Table 1. Tanenbaum96 similarly identified a goal of system design for better performance was to avoid unnecessary copying ("[a packet] is copied to a network layer buffer, then to a transport layer buffer, and finally to the receiving application process.") Ex.1003,

¶129; Ex.1006, .579, .582. Thia's fast-path processing and DMA support bulk data transfer, eliminating the unnecessary copying to multiple buffers associated with the network and transport layers.

Ex.1015, Table 1, .002, .006-.007.

10.3. Claim 6

10.3.1. [6] The method of claim 1, including comparing, by the network interface, the IP addresses and TCP ports of the packets with the source and destination IP addresses and source and destination TCP ports that define the TCP connection.

Thia in combination with Tanenbaum96 in further combination with Stevens2 discloses the method of claim 1 (*See* Section 10.1) and this limitation.

As explained above in Section 9.1, Thia's NIA (part of the *network interface*) receives packets and performs the receive bypass test (*comparing*). Thia also discloses that "The receive bypass test matches the incoming PDU headers with a template that identifies the predicted bypassable headers." Ex.1015, .003.

Similarly in Tanenbaum96, the first step of the fast path test involves checking the connection record against the incoming Transport Protocol Data Units (TPDUs) (i.e., packets) as part of the check to determine whether the received TPDUs were expected. Specifically, this check includes comparing the source and destination *IP addresses* and *TCP ports of the packets* against the connection record. Ex.1006, .584-.585 ("Step 1 is locating the connection record for the incoming TPDU. . . Once

the connection record has been located, both addresses and both ports must be compared to verify that the correct record has been found.") *See also* Section 10.1.3.2 regarding the TCP/IP connection being defined by source and destination IP addresses and source and destination TCP ports. Ex.1003, ¶114.

In combining Thia with Tanenbaum96 for TCP/IP offload, a POSA would have been motivated to use a template for the comparison, such as the prototype header provided by Tanenbaum96, for the receive bypass test. *Id.* ¶128. This template includes source and destination IP addresses and source and destination TCP Ports.

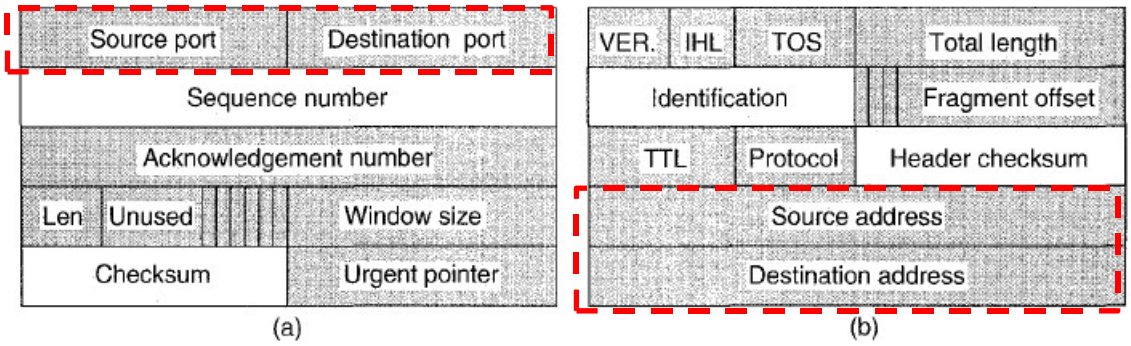


Fig. 6-50. (a) TCP header. (b) IP header. In both cases, the shaded fields are taken from the prototype without change.

Ex.1006, Fig. 6-50 (annotated).

10.4. Claim 7

10.4.1. [7] The method of claim 1, wherein checking whether the packets have certain exception conditions includes checking whether the packets have a RST flag set.

Thia in combination with Tanenbaum96 in further combination with Stevens2 discloses the method of claim 1 (*See* Section 10.1) and this limitation.

As explained in Section 10.1.5.2, Thia's NIA (part of the *network interface*) checks the packets for certain exception conditions as part of its receive bypass test. As explained in Sections 9.4.1 and 9.4.2, a POSA would have been motivated to implement the Jacobson-based Header Prediction disclosed in Tanenbaum96 and Stevens2 as a part of Thia's receive bypass test, which is also based on Jacobson's disclosure. Ex.1003, ¶128. As explained in Section 9.3, Jacobson's Header Prediction test determines whether the RST flag is set. Ex.1013, .962 ("The following four control flags must not be on: SYN, FIN, RST, or URG.")

10.5. Claim 8

10.5.1. [8] The method of claim 1, wherein checking whether the packets have certain exception conditions includes checking whether the packets have a SYN flag set.

Thia in combination with Tanenbaum96 in further combination with Stevens2 discloses the method of claim 1 (*See* Section 10.1) and this limitation.

As explained in Section 10.1.5.2, Thia's NIA (part of the *network interface*) checks the packets for certain exception conditions as part of its receive bypass test.

As explained in Sections 9.4.1 and 9.4.2, a POSA would have been motivated to implement the Jacobson-based Header Prediction as disclosed in Tanenbaum96 and Stevens2 as a part of Thia's receive bypass test, which is also based on Jacobson's disclosure. Ex.1003, ¶128. As explained in Section 9.3, Jacobson's Header Prediction test determines whether the SYN flag is set. Ex.1013, .962 ("The following four control flags must not be on: SYN, FIN, RST, or URG.")

10.6. Claim 9

10.6.1. [9.P] A method for network communication by a host computer having a network interface that is connected to the host by an input/output bus, the method comprising:

As explained in Section 10.1.1 for the identical preamble of Claim 1, to the extent that the preamble is limiting, Thia in combination with Tanenbaum96 in further combination with Stevens2 discloses a method for network communication by a host computer having a network interface that is connected to the host by an input/output bus.

10.6.2. [9.1] receiving, by the network interface, a first packet having a header including source and destination Internet Protocol (IP) addresses and source and destination Transmission Control Protocol (TCP) ports;

Thia in combination with Tanenbaum96 in further combination with Stevens2 discloses this limitation.

As explained above in Section 10.1.1, Thia's NIA, together with the ROPE chip, is the *network interface*. As explained above in Section 9.1, the NIA receives consecutive packets. A network interface connected to a TCP/IP network (as explained above in Section 10.1.3.1) receives TCP/IP packets. Ex.1003, ¶24.

As explained above in Section 10.1.4.2, TCP/IP packets have TCP headers. Therefore, the first received packet has a TCP header.

As explained above in Section 10.3.1, a TCP/IP packet header contains Source and Destination IP addresses and Source and Destination Ports.

Ex.1006, Fig. 6-50.

10.6.3. [9.2] protocol processing, by the host computer, the first packet, thereby initializing a TCP connection that is defined by the source and destination IP addresses and source and destination TCP ports;

Thia in combination with Tanenbaum96 in further combination with Stevens2 discloses this limitation.

As explained above in Section 10.1.2, Thia in combination with Tanenbaum96 in further combination with Stevens2 discloses a protocol processing stack including an IP layer and a TCP layer running on the host computer, namely the modified SPS.

As explained above in Section 10.1.3.1, Thia in combination with Tanenbaum96 in further combination with Stevens2 discloses implementing

Jacobson's Header Prediction as part of Thia's receive bypass test to determine whether a packet is processed on the ROPE chip (fast-path) or on the SPS (slow-path). A packet containing a SYN flag to initialize a connection fails such a receive bypass test because it is not in the ESTABLISHED state and is protocol processed by the SPS on the host (*protocol processing, by the host computer, the first packet, thereby initializing a TCP connection*). Ex.1003, ¶36.

Ex.1006, Fig. 6-28

As explained above in Section 10.1.3.2, a TCP/IP connection is defined by the IP address and TCP port for both sides of the connection.

10.6.4. [9.3] receiving, by the network interface, a second packet having a second header and payload data, wherein the second header has IP addresses and TCP ports that match the IP addresses and TCP ports of the TCP connection;

Thia in combination with Tanenbaum96 in further combination with Stevens2 discloses this limitation.

The first packet in limitation [1.3] discussed in Section 10.1.4 above corresponds to the second packet in this limitation [9.3] because Claim 9 introduces an earlier packet as part of the initialization (*See* Section 10.6.4). As explained above in Section 10.1.4.1, Thia's NIA receives packets and is part of *the network interface*. As explained above in Section 10.1.4.2, modifying Thia to receive TCP/IP packets, the received TCP/IP packets contain headers and optionally contain payload data.

This limitation [9.3] additionally requires that the packet header has IP addresses and TCP ports that match those of the TCP connection. As explained above in Section 4.B, when a connection is initialized, a connection record is recorded containing the source and destination IP addresses and source and destination TCP ports for that connection. As explained above in Section 10.3.1, Thia's modified receive bypass test searches for the matching connection record for each incoming packet using the source and destination IP addresses and source and destination TCP ports stored in each packet. Each packet is compared against a template (connection record) to verify that the correct record has been found. Ex.1003, ¶128.

10.6.5. [9.4] receiving, by the network interface, a third packet having a third header and additional payload data, wherein the third header has IP addresses and TCP ports that match the IP addresses and TCP ports of the TCP connection;

Thia in combination with Tanenbaum96 in further combination with Stevens2 discloses this limitation.

The limitations of [9.4] are identical to [9.3] in Section 10.6.4 (second packet), but for them being in the context of a third packet. A TCP network interface is designed to receive multiple TCP packets for the same TCP connection (e.g. data transfer phase in Thia and Tanenbaum). Ex.1003, ¶¶100, 108. Thus, the disclosures

for the limitations explained above in Section 10.6.4 also disclose this limitation for the same reasons.

10.6.6. [9.5] checking, by the network interface, whether the second and third packets have certain exception conditions, including checking whether the packets are IP fragmented, checking whether the packets have a FIN flag set, and checking whether the packets are out of order;

The limitations [9.5] are substantially identical to [1.4] in Section 10.1.5 as both relate to checking multiple packets; [1.4] checks "the packets" whereas [9.5] checks "the second and third packets" for the same exception conditions. As explained above in Sections 10.1.5.1 and 10.1.5.2, Thia's NIA (part of the *network interface*) receives packets and performs the receive bypass test, which checks whether the packets (including the *second and third packets*) have the claimed exception conditions (*IP fragmented, FIN flag, out of order*).

10.6.7. [9.6] if the second packet has any of the exception conditions, then protocol processing the second packet by the host computer;

Thia in combination with Tanenbaum96 in further combination with Stevens2 discloses if the second packet has any of the exception conditions, then protocol processing the second packet by the host computer.

As explained above in Section 10.6.4, the first packet in claim 1 corresponds to the second packet in claim 9 (*see* Section 10.6.4) because claim 9 introduces an earlier first packet as part of the initialization. As explained in Section 10.1.6,

packets with any of the exception conditions fail Thia's receive bypass test and are protocol processed by the SPS (*by the host computer*).

10.6.8. [9.7] if the third packet has any of the exception conditions, then protocol processing the third packet by the host computer;

As explained above in Section 10.1.7, the same receive bypass test in Thia applies to all packets. *See* Section 10.6.7 above.

10.6.9. [9.8] if the second and third packets do not have any of the exception conditions, then storing the payload data of the second and third packets together in a buffer of the host computer, such that the payload data is stored in the buffer in order and without any TCP header stored between the payload data of the second and third packets.

The limitations of [1.7] in Section 10.1.8 include the limitations in [9.8]. In both sets of limitations, the payload data of the packets without exception conditions are stored in order and without TCP headers between them in a buffer of the host computer. Limitations in [1.7] have an additional limitation of bypassing host protocol processing of the TCP headers, which is not found in [9.8]. Regardless, the disclosures satisfying [1.7] also satisfy [9.8] for the same reasons. *See* Sections 10.1.8.1, 10.1.8.2, and 10.1.8.3.

10.7. Claim 11

10.7.1. [11] The method of claim 9, wherein storing the payload data of the second and third packets together in a buffer of the host computer is performed by a direct memory access (DMA) unit of the network interface.

Thia in combination with Tanenbaum96 in further combination with Stevens2 discloses the method of claim 9 (*see* Section 10.6), wherein storing the payload data of the second and third packets together in a buffer of the host computer is performed by a direct memory access (DMA) unit of the network interface.

This claim is substantially similar to claim 3 (*see* Section 10.2). Claim 3 requires "storing the first payload data and the second payload data together" whereas this claim requires "storing the payload data of the second and third packets together." As previously noted, the first and second packets of claim 1 correspond to the second and third packets of claim 9. *See* Section 10.2.1.

10.8. Claim 14

10.8.1. [14] The method of claim 9, including comparing, by the network interface, the IP addresses and TCP ports of the second and third packets with the source and destination IP addresses and source and destination TCP ports that define the TCP connection.

Thia in combination with Tanenbaum96 in further combination with Stevens2 discloses the method of claim 9 (*see* Section 10.6), including comparing, by the network interface, the IP addresses and TCP ports of the second and third packets

with the source and destination IP addresses and source and destination TCP ports that define the TCP connection.

This claim is substantially similar to claim 6 (*see* Section 10.3). Claim 6 requires "comparing . . . the IP addresses and TCP ports of the packets" wherein packets refer to the first and second packets. This claim requires "comparing . . . the IP addresses and TCP ports of the second and third packets." As previously noted, the first and second packets of claim 1 correspond to the second and third packets of claim 9. *See* Section 10.3.1.

10.9. Claim 15

10.9.1. [15] The method of claim 9, wherein checking whether the second and third packets have certain exception conditions includes checking whether the packets have a RST flag set.

Thia in combination with Tanenbaum96 in further combination with Stevens2 discloses the method of claim 9 (*see* Section 10.6), wherein checking whether the second and third packets have certain exception conditions includes checking whether the packets have a RST flag set.

This claim is substantially similar to claim 7 (*see* Section 10.4). Claim 7 requires "wherein checking whether **the packets** have certain exception conditions includes checking whether the packets have a RST flag set," wherein "the packets" refers to the first and second packets. This claim requires "wherein checking whether **the second and third packets** have certain exception conditions includes

checking whether the packets have a RST flag set." As previously noted, the first and second packets of claim 1 correspond to the second and third packets of claim 9. *See* Section 10.4.1.

10.10. Claim 16

10.10.1. [16] The method of claim 9, wherein checking whether the second and third packets have certain exception conditions includes checking whether the packets have a SYN flag set.

Thia in combination with Tanenbaum96 in further combination with Stevens2 discloses the method of claim 9, wherein checking whether the second and third packets have certain exception conditions includes checking whether the packets have a SYN flag set.

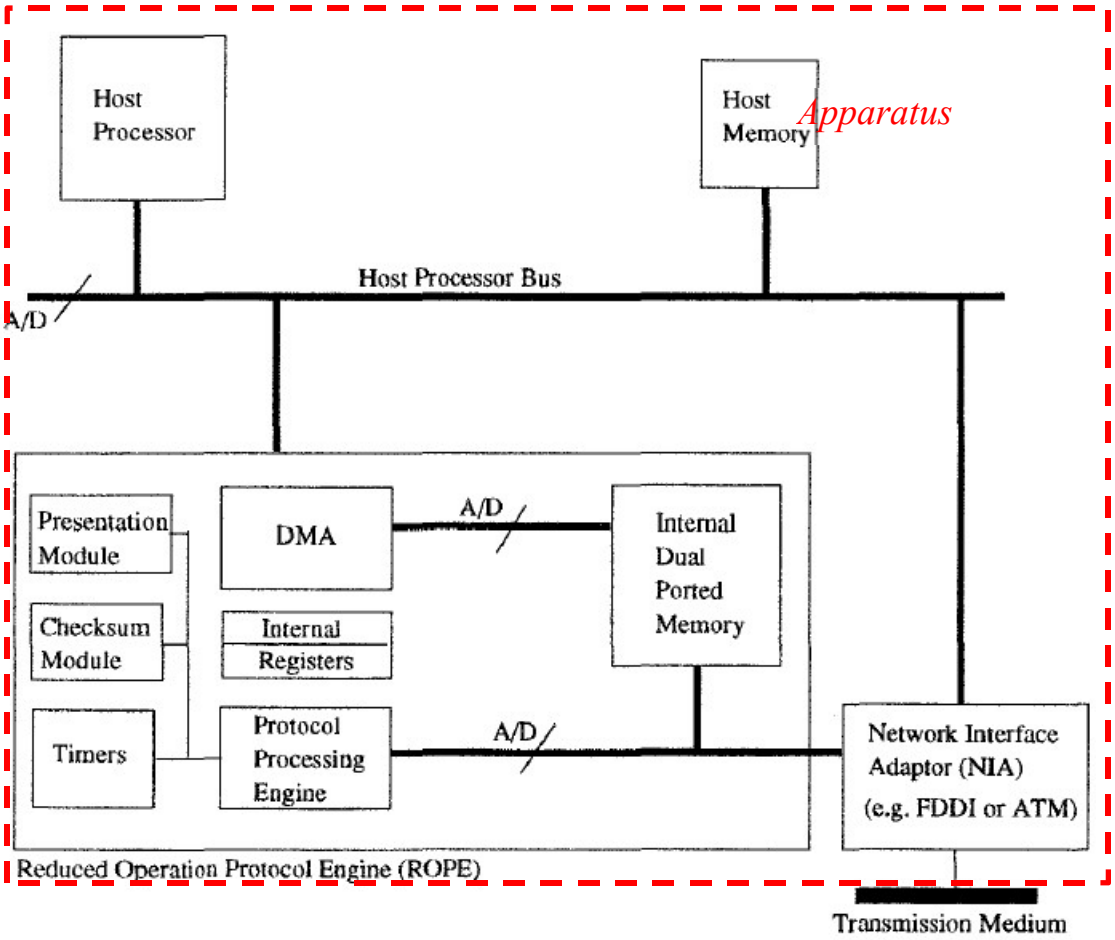
This claim is substantially similar to claim 8 (*see* Section 10.510.6). Claim 8 requires "wherein checking whether **the packets** have certain exception conditions includes checking whether the packets have a SYN flag set," wherein "the packets" refers to the first and second packets. This claim requires "wherein checking whether **the second and third packets** have certain exception conditions includes checking whether the packets have a SYN flag set." As previously noted, the first and second packets of claim 1 correspond to the second and third packets of claim 9. S*ee* Section 10.5.1.

10.11. Claim 17

10.11.1. [17.P] An apparatus for network communication, the apparatus comprising:

To the extent that the preamble is limiting, Thia in combination with Tanenbaum96 in further combination with Stevens2 discloses an apparatus for network communication.

Thia discloses an apparatus for network communication. Specifically, Thia discloses a Reduced Operation Protocol Engine (ROPE) chip and a Network Interface Adaptor (NIA) connected to the Host Processor and Host Memory by a Host Processor Bus (together the "*apparatus*"). The *apparatus* is connected to the network through the Transmission Medium.



Ex.1015, .007 (annotated).

Thia discloses that the NIA (portion of the *apparatus*) is used for sending and receiving packets (*network communications*). Ex.1015, .008 (describing the NIA as a source/sink for data packets.)

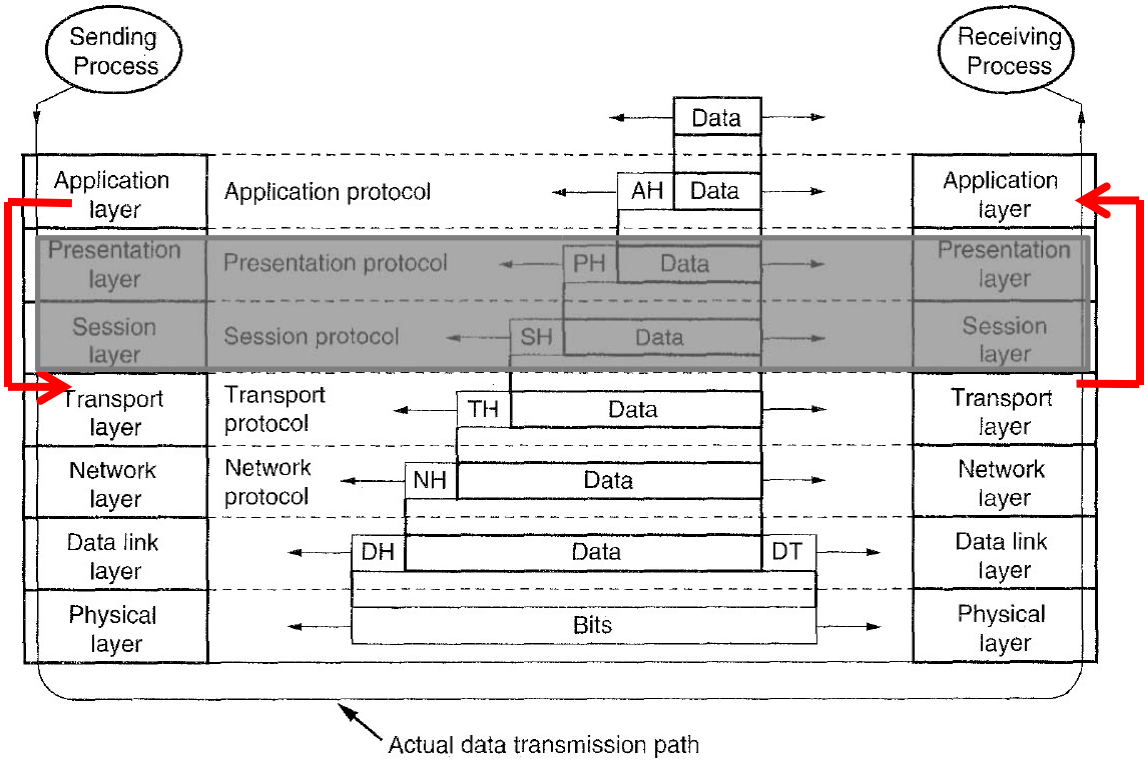
10.11.2. [17.1] a host computer running a protocol stack including an Internet Protocol (IP) layer and a Transmission Control Protocol (TCP) layer, the protocol stack adapted to establish a TCP connection for an application layer running above the TCP layer, the TCP connection being defined by source and destination IP addresses and source and destination TCP ports;

Claim 1 is a method claim. Claim 17 is an apparatus claim. The evidence and disclosures from the following Sections for claim 1 also disclose the limitations of the claimed apparatus. *See* Section 10.1.

As explained above in Sections 10.1.2.1 and 10.1.2.2, Thia in combination with Tanenbaum96 in combination with Stevens2 discloses an SPS running on the host (*a host computer running a protocol stack*). As explained above in Sections 10.1.2.3, and 10.1.2.4, Thia in combination with Tanenbaum96 in combination with Stevens2 teach modifying the SPS to implement the TCP/IP protocol suite, which includes an IP layer and a TCP layer with an application layer running above the TCP layer (*including an Internet Protocol (IP) layer and a Transmission Control Protocol (TCP) layer*).

As explained above in Section 10.1.3.1, Thia in combination with Tanenbaum96 in combination with Stevens2 discloses that the SYN packets required for the initialization fail the conditions of the receive bypass test, and thus must be processed by the SPS on the host. Accordingly, the SPS is responsible for

initializing the TCP connection for the application layer running above the TCP layer (*the protocol stack adapted to establish a TCP connection for an application running above the TCP layer*).



Ex.1006, Fig. 1-17 (annotated).

As explained above in Section and 10.1.3.2, Thia in combination with Tanenbaum96 in combination with Stevens2 discloses that the TCP connection is defined by the TCP ports and the IP addresses for both sides of the connection (*the TCP connection being defined by source and destination IP addresses and source and destination TCP ports*).

10.11.3. [17.2] a network interface that is connected to the host computer by an input/output bus, the network interface adapted to parse the headers of received packets to determine whether the headers have the IP addresses and TCP ports that define the TCP connection and to check whether the packets have certain exception conditions, including whether the packets are IP fragmented, have a FIN flag set, or are out of order, the network interface having logic that directs any of the received packets that have the exception conditions to the protocol stack for processing, and directs the received packets that do not have any of the exception conditions to have their headers removed and their payload data stored together in a buffer of the host computer, such that the payload data is stored in the buffer in order and without any TCP header stored between the payload data that came from different packets of the received packets.

As shown by the following sections, Thia in combination with Tanenbaum96 in further combination with Stevens2 discloses this limitation.

10.11.3.1. a network interface that is connected to the host computer by an input/output bus

As explained above in Section 10.1.1, Thia in combination with Tanenbaum96 in further combination with Stevens2 discloses a host computer having a network interface that is connected to the host by an input/output bus. Ex.1015, Fig. 2.

10.11.3.2. the network interface adapted to parse the headers of received packets

A POSA would have understood that parsing a header involves identifying fields of the header to determine whether or not those fields individually or collectively satisfy one or more criteria. Ex.1003, ¶102. Thia discloses a receive bypass test (performed by the NIA, part of *the network interface*) in which fields of the header of a received packet are compared to a template. Ex.1015, .003 ("The receive bypass test matches the incoming PDU headers with a template that identifies the predicted bypassable headers.") A POSA would have understood that before the NIA compares fields within a header, the header must be parsed. Ex.1003, ¶102.

10.11.3.3. to determine whether the headers have the IP addresses and TCP ports that define the TCP connection and

As explained in Section 10.3.1, Thia in combination with Tanenbaum96 in further combination with Stevens2 discloses the NIA performing a modified receive bypass test that checks whether the packet has source and destination IP addresses and source and destination TCP ports that match the connection record. In doing so, the NIA determines *whether the headers have the IP addresses and TCP ports that define the TCP connection*. *Id*. ¶130.

10.11.3.4. to check whether the packets have certain exception conditions, including whether the packets are IP fragmented, have a FIN flag set, or are out of order

As explained in Sections 10.1.5.1 and 10.1.5.2, the NIA (part of *the network interface*) performs a modified receive bypass test as taught by Tanenbaum96 on received packets to check whether each TCP header has an established connection, neither side is trying to close the connection (i.e., *no FIN or RST flag*), the TPDU is a full one (e.g*., no IP fragmentation*), no special flags are set (e.g., *no FIN, RST or SYN flag*), and the sequence number is the one expected (i.e., *the packets are not out of order*). *See* Ex.1006, .567 (disclosing the test above); *see also* Ex.1013, .936-.937 (providing a walkthrough of the BSD code for the test above).

10.11.3.5. the network interface having logic that directs any of the received packets that have the exception conditions to the protocol stack for processing

As explained above in Section 10.1.6, Thia in combination with Tanenbaum96 in further combination with Stevens2 discloses that packets that fail the bypass test are processed by the SPS.

10.11.3.6. and directs the received packets that do not have any of the exception conditions to have their headers removed and their payload data stored together in a buffer of the host computer, such that the payload data is stored in the buffer in order and without any TCP header stored between the payload data that came from different packets of the received packets.

As explained above in Section 10.1.8.1, Thia in combination with Tanenbaum96 in further combination with Stevens2 discloses that packets that pass the receive bypass test (*packets that do not have any of the exception conditions*) are bypassed to the ROPE chip for processing.

Ex.1015, Fig. 1.

As explained above in Section 10.1.8.2, Thia in combination with Tanenbaum96 in further combination with Stevens2 discloses that the system uses DMA to move only packet data in order from the ROPE chip to host memory (*to have . . . their payload data stored together in a buffer of the host computer*).

Ex.1015, Thia Fig. 2.

As explained above in Section 10.1.8.3, a POSA would have appreciated that a packet passing the receive bypass test has a header decoded by the ROPE chip, and thus there is no need to transfer the already-decoded header into host memory (*to have their header removed*). Additionally, as explained in that same section, the

TCP entity on the ROPE chip reconstructs the data from each packet into the original byte stream (removing the headers from each datagram and arranging them in order).

Thus, a POSA would have been motivated to develop a TCP *network interface* combining Thia's protocol engine with the TCP disclosures found in Tanenbaum96 and Stevens2. As explained above, when incoming packets pass the modified receive bypass test (*the received packets that do not have any of the exception conditions*), they are directed to be processed by the ROPE chip as opposed to the SPS. The output from the ROPE chip is the original byte stream (*to have their headers removed . . . in order and without any TCP header stored between the payload data that came from different packets of the received packets*). A POSA would have utilized Thia's on-chip Direct Memory Access (DMA) to transfer only the data blocks in order to a buffer in host memory (*and their payload data stored together in a buffer of the host computer, such that the payload data is stored in the buffer*). Ex.1003, ¶128.

10.12. Claim 19

10.12.1. [19] The apparatus of claim 17, wherein the network interface includes a direct memory access (DMA) unit that is adapted to store the payload data in the buffer.

Thia in combination with Tanenbaum96 in further combination with Stevens2 discloses the apparatus of claim 17 (*see* Section 10.11), wherein the network

interface includes a direct memory access (DMA) unit that is adapted to store the payload data in the buffer.

Claim 1 is a method claim. Claim 19 depends from claim 17, which is an apparatus claim. The evidence and disclosures from the following Sections for claim 1 also disclose the limitations of the claimed apparatus. *See* Section 10.1.

As explained in Section 10.1.8.2, Thia teaches the use of DMA to move packet data from the ROPE chip to host memory (a host buffer).

10.13. Claim 21

10.13.1. [21] The apparatus of claim 17, wherein the exception conditions include having a RST flag set.

Thia in combination with Tanenbaum96 in further combination with Stevens2 discloses the apparatus of claim 17 (*see* Section 10.11), wherein the exception conditions include having a RST flag set.

Claims 1 and 7 are method claims. Claims 17 and 21 are apparatus claims. The evidence and disclosures described above in Section 10.4.1 disclosing an exception condition including a RST flag set for claim 7 also disclose the limitations of this claim.

10.14. Claim 22

10.14.1. [22] The apparatus of claim 17, wherein the exception conditions include having a SYN flag set.

Thia in combination with Tanenbaum96 in combination with Stevens2 discloses the apparatus of claim 17 (*see* Section 10.11), wherein the exception conditions include having a SYN flag set.

Claims 1 and 8 are method claims. Claims 17 and 22 are apparatus claims. The evidence and disclosures described above in Section 10.5.1 showing an exception condition that includes having a SYN flag set meeting claim 8 also disclose the limitations of this claim.

11. CONCLUSION

For the reasons set forth above, *inter partes* review of claims 1, 3, 6-9, 11, 14-17, 19, 21, and 22 of the 948 Patent is requested.

Respectfully submitted,

DUANE MORRIS LLP

Dated: July 1, 2017

By: */Patrick D. McPherson/*

Patrick D. McPherson,
Reg. No. 46,255
505 9th Street, N.W., Suite 1000
Washington, D.C. 20004
P: (202) 776-7800
F: (202) 776-7801
PDMcPherson@duanemorris.com

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 13,912 words as calculated by the "Word Count" feature of Microsoft Word 2010, the word processing program used to create it.

DUANE MORRIS LLP

Dated: July 1, 2017

By: */Patrick D. McPherson/*

Patrick D. McPherson,
Reg. No. 46,255
505 9th Street, N.W., Suite 1000
Washington, D.C. 20004
P: (202) 776-7800
F: (202) 776-7801
PDMcPherson@duanemorris.com

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 July 1, 2017

Manner of service EXPRESS MAIL

Documents served Petition for *Inter Partes* Review of U.S. Pat. No. 8,805,948 with Exhibit List
Power of Attorney
Exhibits Ex.1001 - Ex.1064

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

Dated: July 1, 2017

/Patrick D. McPherson/
Patrick D. McPherson
Lead Counsel for Petitioners
Registration No. 46,255