

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. 8,850,948
Title: INTELLIGENT NETWORK INTERFACE SYSTEM AND METHOD FOR
PROTOCOL PROCESSING

**Declaration of Robert Horst, Ph.D. in Support of
Petition for *Inter Partes* Review
of U.S. Patent No. 8,850,948**

TABLE OF CONTENTS

	Page
I. INTRODUCTION AND QUALIFICATIONS	1
II. MATERIALS RELIED ON IN FORMING MY OPINION.....	3
III. UNDERSTANDING OF THE GOVERNING LAW	4
A. Invalidity by Anticipation	4
B. Invalidity by Obviousness	5
IV. LEVEL OF ORDINARY SKILL IN THE ART	6
V. STATE OF THE ART AND OVERVIEW OF TECHNOLOGY AT ISSUE.....	8
A. Layered Network Protocols.....	8
1. OSI Layers	8
2. TCP/IP Layers.....	8
B. TCP/IP	10
1. Encapsulation	12
2. Ethernet Header.....	14
3. IP Header.....	16
4. TCP header.....	17
5. Application Data	22
6. RFC 793 – TCP Specification.....	22
B. Protocol Offload and Fast-Path Processing.....	22
1. RFC 647 – Front-Ending.....	23
2. RFC 929 – Outboard Processing.....	24
3. Mediation Levels.....	25
C. Offloaded Protocols.....	28
1. OSI Protocol Offload	28
2. TCP/IP Protocol Offload.....	28
3. VMTP and XTP Protocol Offload	28
4. Multi-Protocol Offload	29

D.	Portions of the Protocol Offloaded.....	29
1.	Checksum Offload	30
2.	Full Offload.....	30
3.	Multi-Level Offload	31
4.	Header Prediction.....	31
E.	Offload Implementation	34
1.	Multiprocessor Offload	34
2.	Offload Adapters based on Microprocessors	36
3.	Offload Adapters based on Custom Processors or Custom Logic	37
F.	Protocol Offload Summary	40
G.	Additional Background Technology	41
1.	DMA	41
2.	Virtual and Physical Memory Addresses.....	43
VI.	OVERVIEW OF 948 PATENT	45
VII.	948 PATENT PROSECUTION HISTORY	48
VIII.	CLAIM CONSTRUCTIONS	49
A.	Legal Standard.....	49
IX.	THE PRIOR ART.....	50
A.	Thia: Thia, A Reduced Operation Protocol Engine (ROPE) for a multiple-layer bypass architecture (1995).....	50
B.	Tanenbaum96: A. Tanenbaum, Computer Networks, 3rd ed. (1996)	57
C.	Stevens2: Stevens, TCP-IP Illustrated, Vol. 2	68
X.	Obviousness Combinations – Motivations To Combine.....	71
A.	Thia in Combination with Tanenbaum96.....	71
B.	Thia in Combination with Tanenbaum96 and further in Combination with Stevens2	75
XI.	GROUND OF INVALIDITY	77

I, Robert Horst, hereby declare as follows:

I. INTRODUCTION AND QUALIFICATIONS

1. My name is Robert Horst. I have been retained on behalf of Petitioner Cavium, Inc. (“Cavium”) to provide this Declaration concerning technical subject matter relevant to the petition for *inter partes* review (“Petition”) concerning U.S. Patent No. 8,850,948 (Ex.1001, the “948 Patent”). I reserve the right to supplement this Declaration in response to additional evidence that may come to light.

2. I am over 18 years of age. I have personal knowledge of the facts stated in this Declaration and could testify competently to them if asked to do so.

3. My compensation is not based on the resolution of this matter. My findings are based on my education, experience, and background in the fields discussed below.

4. I am an independent consultant with more than 30 years of expertise in the design and architecture of computer systems. My current curriculum vitae is submitted as Exhibit 1004 and some highlights follow.

5. Currently, I am an independent consultant at HT Consulting where my work includes consulting on technology and intellectual property. I have testified as an expert witness and consultant in patent and intellectual property litigation as well as *inter partes* reviews and re-examination proceedings.

6. I earned my M.S. (1978) in electrical engineering and Ph.D. (1991) in computer science from the University of Illinois at Urbana-Champaign after earning my B.S. (1975) in electrical engineering from Bradley University. During my master’s program, I designed, constructed and debugged a shared memory parallel microprocessor system. During my doctoral program, I designed and simulated a massively parallel, multi-threaded task flow computer.

7. After receiving my bachelor’s degree and while pursuing my master’s degree, I worked for Hewlett-Packard Co. While at Hewlett-Packard, I designed the micro-sequencer and cache of the HP3000 Series 64 processor. From 1980 to 1999, I worked at Tandem Computers, which was acquired by Compaq Computers in 1997. While at Tandem, I was a designer and architect of several generations of fault-tolerant computer systems and was the principal architect of the NonStop Cyclone superscalar processor. The system development work at Tandem also included development of the ServerNet System Area Network and applications of this network to fault tolerant systems and clusters of database servers.

8. Since leaving Compaq in 1999, I have worked with several technology companies, including 3Ware, Network Appliance, Tibion, and AlterG in the areas of network-attached storage and biomedical devices. From 2012 to 2015, I was Chief Technology Officer of Robotics at AlterG, Inc., where I worked

on the design of anti-gravity treadmills and battery-powered orthotic devices to assist those with impaired mobility.

9. In 2001, I was elected an IEEE Fellow “for contributions to the architecture and design of fault tolerant systems and networks.” I have authored over 30 publications, have worked with patent attorneys on numerous patent applications, and I am a named inventor on 80 issued U.S. patents.

10. My patents include those directed to networks (e.g., U.S. Pat. No. 6,157,967: Method of data communication flow control in a data processing system using busy/ready commands), storage (e.g., U.S. Pat. No. 6,549,977: Use of deferred write completion interrupts to increase the performance of disk operations), and multi-processor systems (e.g., U.S. Pat. No. 5,751,932: Fail-fast, fail-functional, fault-tolerant multiprocessor system). My publications include a conference paper that examined the performance and efficacy of protocol offload engines (Ex.1004).

11. My Curriculum Vitae, which is filed as a separate Exhibit (Ex.1004), contains further details on my education, experience, publications, and other qualifications to render this opinion as expert.

II. MATERIALS RELIED ON IN FORMING MY OPINION

12. In addition to reviewing U.S. Patent No. 8,850,948 (Ex.1001), I also reviewed and considered the prosecution history of the 948 Patent (Ex.1002). I

also reviewed Thia, A ROPE for multiple-layer bypass architecture (“Thia”) (Ex.1015), A. Tanenbaum, 3rd ed. (1996) (Ex.1006), and Stevens, TCP-IP Illustrated, Vol.2 (“Stevens2”) (Ex.1013). I also considered the background materials cited herein.

III. UNDERSTANDING OF THE GOVERNING LAW

13. I understand that a patent claim is invalid if it is anticipated or rendered obvious in view of the prior art. I further understand that invalidity of a patent claim requires that the claim be anticipated or obvious from the perspective of a person of ordinary skill in the relevant art at the time the invention was made.

A. Invalidity by Anticipation

14. I have been informed that a patent claim is invalid as anticipated under 35 U.S.C. § 102 if each and every element of a claim, as properly construed, is found either explicitly or inherently in a single prior art reference.

15. I have been informed that a claim is invalid under 35 U.S.C. § 102(a) if the claimed invention was patented or published anywhere, before the applicant's invention. I further have been informed that a claim is invalid under 35 U.S.C. § 102(b) if the invention was patented or published anywhere more than one year prior to the first effective filing date of the patent application (critical date). I further have been informed that a claim is invalid under 35 U.S.C. § 102(e) if an invention described by that claim was disclosed in a U.S. patent granted on an

application for a patent by another that was filed in the U.S. before the date of invention for such a claim.

B. Invalidity by Obviousness

16. I have been informed that a patent claim is invalid as obvious under 35 U.S.C. § 103 if it would have been obvious to a person of ordinary skill in the art, taking into account (1) the scope and content of the prior art, (2) the differences between the prior art and the claims, (3) the level of ordinary skill in the art, and (4) any so called “secondary considerations” of non-obviousness, which include: (i) “long felt need” for the claimed invention, (ii) commercial success attributable to the claimed invention, (iii) unexpected results of the claimed invention, and (iv) “copying” of the claimed invention by others. I further understand that it is improper to rely on hindsight in making the obviousness determination. I have been informed that Alacritech claims a filing priority date no later than October 14, 1997 for claims 1, 3, 6-9, 11, 14-17, 19, and 21-22 of the 948 Patent. Accordingly my analysis of the prior art for the claims of the 948 Patent is based on the prior art and knowledge of a person having ordinary skill in the art (“POSA”) as of October 14, 1997.

17. I have been informed that a claim can be obvious in light of a single prior art reference or multiple prior art references. I further understand that exemplary rationales that may support a conclusion of obviousness include:

- (A) Combining prior art elements according to known methods to yield predictable results;
- (B) Simple substitution of one known element for another to obtain predictable results;
- (C) Use of known technique to improve similar devices (methods, or products) in the same way;
- (D) Applying a known technique to a known device (method, or product) ready for improvement to yield predictable results;
- (E) “Obvious to try” - choosing from a finite number of identified, predictable solutions, with a reasonable expectation of success;
- (F) Known work in one field of endeavor may prompt variations of it for use in either the same field or a different one based on design incentives or other market forces if the variations are predictable to one of ordinary skill in the art;
- (G) Some teaching, suggestion, or motivation in the prior art that would have led one of ordinary skill to modify the prior art reference or to combine prior art reference teachings to arrive at the claimed invention.

IV. LEVEL OF ORDINARY SKILL IN THE ART

18. I have been informed that factors that may be considered in determining the level of ordinary skill in the art may include: (A) “type of

problems encountered in the art;” (B) “prior art solutions to those problems;” (C) “rapidity with which innovations are made;” (D) “sophistication of the technology;” and (E) “educational level of active workers in the field.” I also understand that, every factor may not be present for a given case, and one or more factors may predominate. Here, the 948 Patent is directed to an apparatus and methods for receive side network protocol offload. In my experience, systems such as those capable of protocol offload are not designed by a single person but instead require a design team with wide ranging skills and experience including computer architecture, network design, software development and hardware development. Moreover, the design team typically would have comprised individuals with advanced degrees and some industry experience, or significant industry experience.

19. Accordingly, and while it would be rare to find all of these skills in a single individual, it is my opinion that a person of ordinary skill in the art (“POSA”) is 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.

20. The statements that I make in this declaration when I refer to a POSA are from the perspective of October 14, 1997.

V. STATE OF THE ART AND OVERVIEW OF TECHNOLOGY AT ISSUE

21. In this section, I provide an overview of the technology at issue and illustrate the state of the art.

A. Layered Network Protocols

22. The primary goal of computer networking is to provide fast, reliable data communications between computer systems. Interoperability has been accomplished through adherence to standards, and performance has steadily increased through new technology and optimizations of hardware and software.

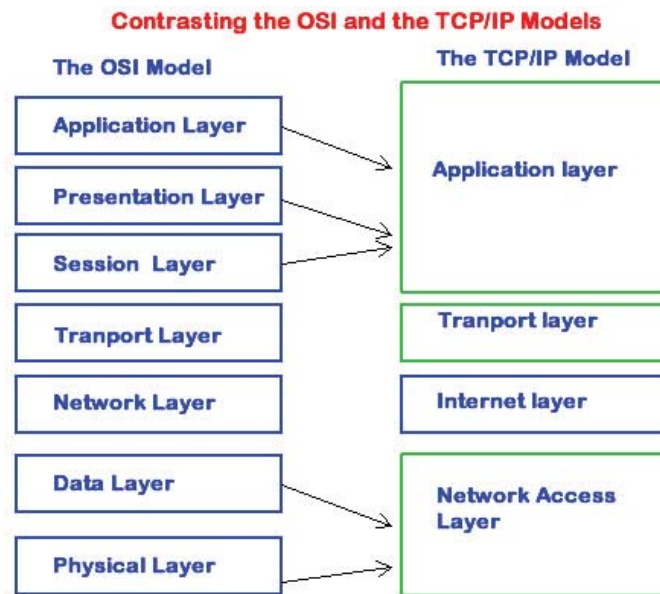
1. OSI Layers

23. Computer networking standards provide inter-system communications across a wide range of hardware and software implementations. The seven-layer OSI model describes a logical layering including physical, data link, network, transport, session, presentation and application as illustrated below.

2. TCP/IP Layers

24. The TCP/IP layering is slightly different and corresponds more closely to the way the networking code is typically partitioned in some popular Unix variants. TCP/IP layers include physical (e.g. 100baseT, 1000baseT), data

link¹ (e.g. IEEE 802 Ethernet, ATM, Token Ring), Internet (e.g. IPv4, IPv6), transport (e.g. TCP, UDP, VMTP, XTP), and Application (e.g. FTP, SMTP, Telnet, HTTP). A network interface connected to a TCP/IP network receives TCP/IP packets that comply with the TCP/IP protocol. The following figure shows the relationship between the OSI and TCP/IP layering.



¹ References on TCP/IP use different terminology to describe the layer under IP layer. The data link layer is also called the “host-to-network layer” in Tanenbaum96 and the “interface layer” in Stevens2 (see below for description of these references). Some Alacritech patents use “data link layer,” “link layer” and “MAC layer.” Prior art references use many of these terms and also sometimes use the name of a specific implementation (e.g. Ethernet, ATM).

Available at <http://mitigationlog.com/how-tcpip-and-reference-osi-model-works/>.²

An application layer is above the transport layer in both protocols.

25. At a conceptual level, each layer is responsible only for its respective functions. This enables, for example, hiding the complexity of the physical data connection (that is, actually transmitting the data onto the physical wires) from layers above the physical, data link, and network layers above. Likewise, the lower layers must transmit the data on the physical wires, but need not worry about what application the data belongs to or even whether it is receiving packets in the correct order.

B. TCP/IP

26. 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, 948 Patent at Abstract. The claims are all directed to TCP/IP.

27. By the mid 1990s, TCP/IP was a firmly entrenched standard and was a widespread networking protocol to, for example, access the Internet and World Wide Web, overtaking the OSI protocols. *See* Ex.1006, Tanenbaum96 at .016

² It appears that this diagram was made in 2012. It is being used for illustrative purposes only.

(“The OSI protocols have quietly vanished, and the TCP/IP protocol suite has become dominant.”). By that time, detailed descriptions of the protocols and open-source implementations were widely available from books, technical papers, and code repositories. Free implementations of TCP/IP, such as Free BSD, were widely available and widely used. Standard reference books on TCP/IP included Stevens1 (Ex.1008), Stevens2 (Ex.1013), and Tanenbaum96 (Ex.1006), all of which were widely cited and relied upon.³ A series of technical memos called RFCs (request for comments) document the progression of design concepts of the Internet. 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, RFC 793; Ex.1036, RFC 791. A few of the key RFCs are quoted below to establish when certain concepts were proposed and documented.

28. 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

³ These books were well known resources to a POSA. Consistent with that, Alacritech patents cite editions of the Tanenbaum and Stevens books.

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 during transmission and reassembly when received. Fragmentation occurs when an IP packet must be divided (“fragmented”) into smaller packets when a packet travels over an intermediate network with a small packet size. TCP network interface includes the ability to receive multiple TCP packets for the same connection. TCP/IP can be transmitted over a variety of physical media (e.g. Ethernet).

1. Encapsulation

29. Network layering corresponds to the encapsulation of higher levels by lower levels. 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.

30. The following figure shows an example with application data accompanied by an application header. 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. The application header-data combination becomes the application data of a TCP segment. The TCP segment containing the application header-data combination along with the IP header forms an IP datagram. The IP datagram along with an appropriate MAC (media access control) layer header forms the frame that is sent over the physical interconnect. The diagram below shows an example of such encapsulation where the MAC layer is Ethernet. Some software implementations implement the layers separately with data, or pointers to data, passed between the software modules for each layer. In this case, one module creates the user data and application header, another module then encapsulates that with a TCP header, etc. The processing occurs sequentially, from top to bottom, as shown below.

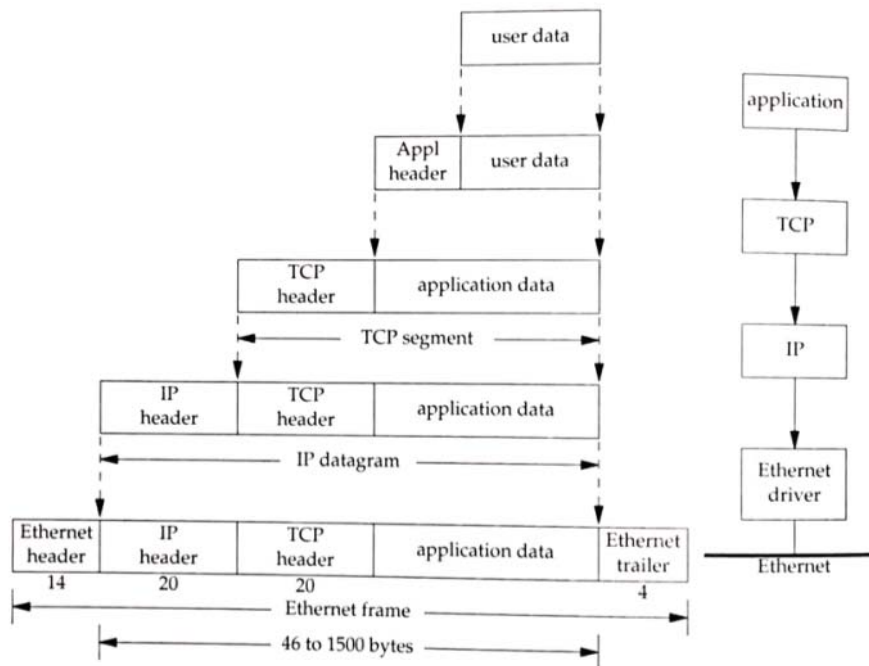
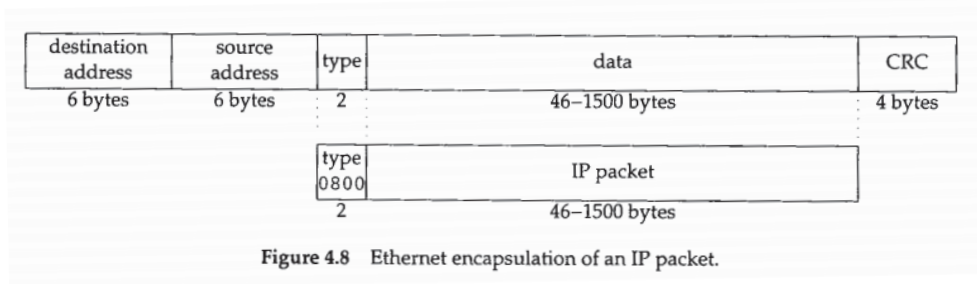


Figure 1.7 Encapsulation of data as it goes down the protocol stack.

Ex.1008, Stevens1 at .034. 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. The user data without headers is eventually delivered to the relevant application.

2. Ethernet Header

31. The lowest layer, the MAC (media access control) layer handles the actual transmission on the physical media. A 14-byte Ethernet header, for example, includes 48-bit (6 byte) source and destination MAC addresses for uniquely identifying the network interface (e.g., on a computer or router) on a local area network at each end of the link.



Ex.1013, Stevens2 at .125.

32. The MAC address can be determined by a routing table in the protocol stack. In an Ethernet-based network, the 48-bit MAC address corresponds to a physical interface, such as a network interface card (NIC) or WiFi modem in a server or router. The MAC address field of the destination in the Ethernet header determines the next hop along the route to the destination. At each router along the path, the MAC address field is changed to the MAC address of the next router. The final router changes the MAC address field to the MAC address of the destination.

3. IP Header

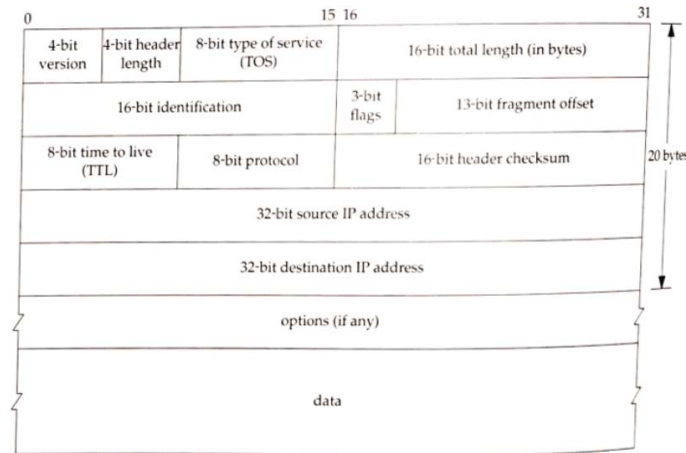


Figure 3.1 IP datagram, showing the fields in the IP header.

Ex.1008, Stevens1 at .058.

33. Above the MAC layer is the Internet protocol layer (IP layer). An IP header is illustrated by the figure above from Stevens1. The IP header includes source and destination IP addresses for identifying the end points (e.g., computer) of the connection. The IP header also has a flag that indicates whether the packet has been fragmented. The 32-bit IPv4 addresses are usually expressed in dotted decimal notation. For example, an IP address of Google.com is 216.58.216.46.

4. TCP header

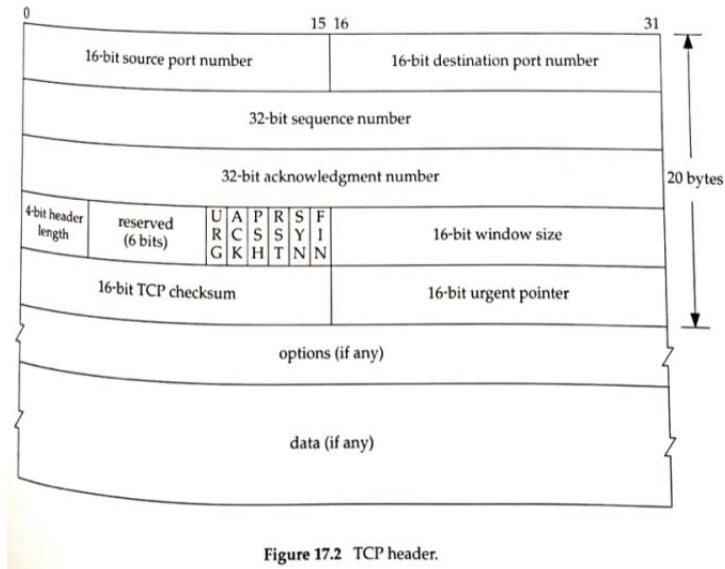


Figure 17.2 TCP header.

Ex.1008, Stevens1 at .249.

34. Above the IP layer is the TCP (Transport) layer. A TCP header is illustrated by the figure above from Stevens1. The TCP header includes 16-bit source and destination port numbers for identifying the processes that are communicating. These port numbers identify the end points (e.g., client or server programs) sending and receiving data on each end of the connection. TCP is used to establish connections between processes at IP addresses across the network and the TCP port numbers identify which processes are communicating. For instance, Email may use SMTP (simple mail transfer protocol) on port 25 (SMTP’s well-known port number) while a web server is using HTTP on port 80 (HTTP’s well-known port number).

35. The TCP layer performs several important functions such as tracking the sequence of packets to ensure that the TCP packets are assembled in the proper order. As shown above, a “sequence number” is included in the TCP header for several reasons such as identifying TCP packets and performing reassembly of these packets. The TCP layer tracks and acknowledges the sequence of packets, so that the sending TCP layer can re-send lost (and therefore unacknowledged) data so that the application does not have 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.

36. TCP maintains the status of each connection with a finite state machine. 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, RFC 793 at .016. The finite state machine is also illustrated in Tanenbaum96 below.

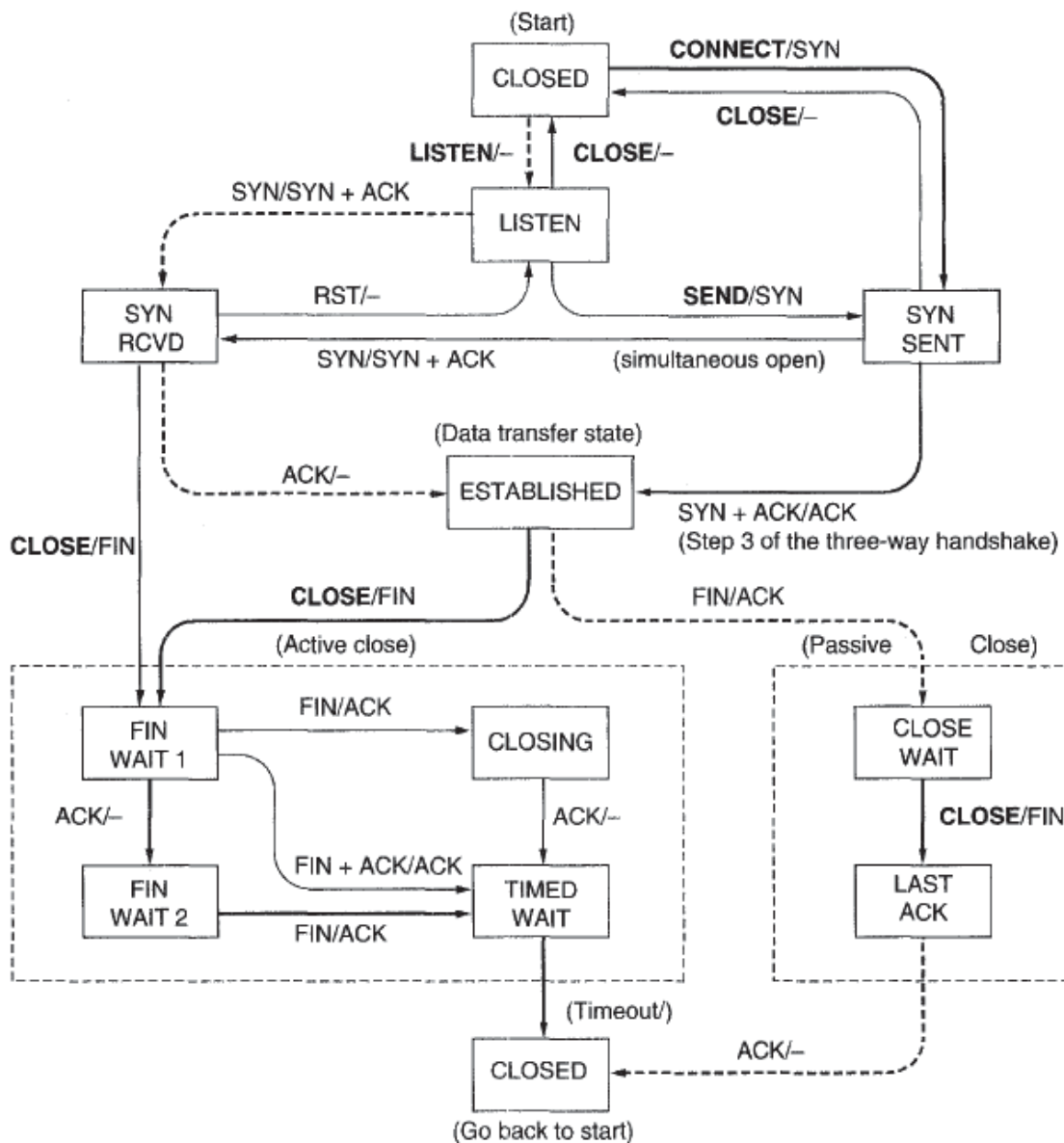


Fig. 6-28. TCP connection management finite state machine. The heavy solid line is the normal path for a client. The heavy dashed line is the normal path for a server. The light lines are unusual events.

Ex.1006, Tanenbaum at .550.

Connections begin in a CLOSED state. Different control flags in the TCP header of packets sent between client and server affect the state of the connection. These control flags are URG, ACK, PSH, RST, SYN, and FIN as illustrated below.

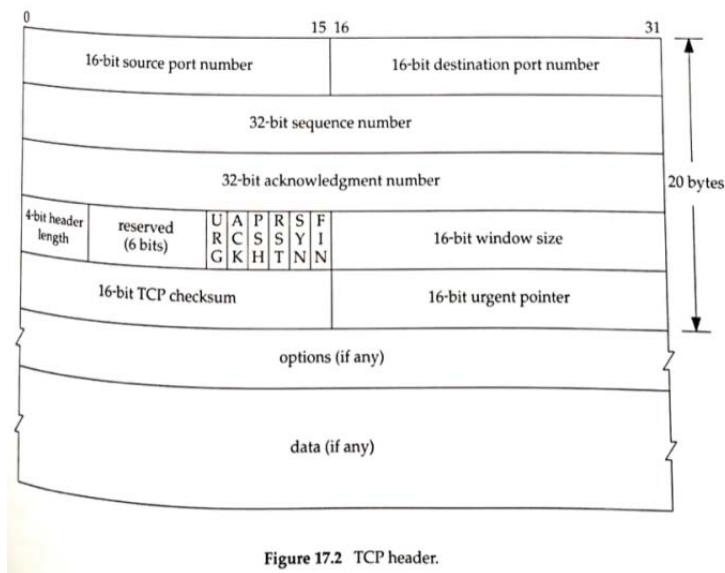


Figure 17.2 TCP header.

Ex.1008, Stevens1 at .249. Certain control flags indicate that the connection is not yet established or will be closed. A server can move from CLOSED into a LISTEN state, where it will wait until a request to initialize a connection is received in a packet with the SYN flag set. A server is not in the ESTABLISHED state until the server acknowledges the SYN packet. In the ESTABLISHED state, data is transferred over the connection. Either side can close the connection by sending a special packet called a FIN packet. Control flags URG, and RST are also requests to close a connection, which indicates that the server is no longer in the ESTABLISHED state.

37. Accordingly, routing packets between source and destination processes over a TCP/IP connection using Ethernet requires TCP source and destination port numbers, source and destination IP addresses, and source and

destination MAC addresses. For more information on TCP, *see* Stevens1 (Ex.1008) Chapter 17, “TCP: Transmission Control Protocol,” .247-252.

5. Application Data

38. Each user application typically has at least one range of addresses in the user space region of host memory where it places data for transmission and receives data from the network. For transmission, the protocol stack can retrieve data from this area in host memory, encapsulate it in packets as described above, and then transmit it over the network. For receipt of data, the protocol stack puts data in the assigned host memory after it has processed and stripped off the MAC, IP, and TCP headers from the packet.

6. RFC 793 – TCP Specification

39. The original TCP specification was published in RFC 793 (Ex.1007) in September 1981. RFC 793 is a full specification for TCP and shows, among many other things, that identifying a TCP connection by its source and destination IP addresses and TCP ports were known more than 15 years before the earliest priority dates of the Alacritech patents.

B. Protocol Offload and Fast-Path Processing

40. To increase performance and reduce demands on the host computer required for protocol processing, designers have employed different techniques such as parallel processing, improved hardware, memory copy reduction via hardware and/or software, and hardware to offload all or part of the protocol stack.

1. RFC 647 – Front-Ending

41. As early as 1974, front-end protocol offload was already being considered for standardization as described in request-for-comments RFC 647. This represents the consensus at the time that front ending (the offloading of protocol processing) was desirable. At that time, NCP (Network Control Protocol) was the protocol used in ARPANET, the predecessor to the modern Internet.

“FRONT-ENDING”

In what might be thought of as the greater network community, the consensus is so broad that the front-ending is desirable that the topic needs almost no discussion here. Basically, a small machine (a PDP-11 is widely held to be most suitable) is interposed between the IMP and the Host in order to shield the Host from the complexities of the NCP.

Ex.1019, RFC 647 at .002.

42. RFC 647 goes on to discuss rigid and flexible front-end (FE) alternatives and includes a high-level discussion of a protocol for interfacing between the host and FE.

2. RFC 929 – Outboard Processing

43. In 1984, RFC 929 was distributed to begin work on a possible standard for interfacing between a host and an OPE (Outboard Processing Environment)⁴:

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.

Ex.1009, RFC 929 at .002.

⁴ Other names have been used to describe the OPE concept. Names for protocol offload implementations included Front-End Processor, Network Front-End, Protocol Processor, Protocol Engine, Protocol Accelerator, Hardware Bypass, Smart Network Interface, SMART NIC, Smart Adapter, Protocol Processing Engine, IO Adapter, Intelligent I/O Processor and intelligent Network Interface Card.

The interaction between the Host and the OPE must be capable of providing a suitable interface between processes (or protocol interpreters) in the Host and the off-loaded protocol interpreters in the OPE. This interaction must not, however, burden the Host more heavily than would have resulted from supporting the protocols inboard, lest the advantage of using an OPE be overridden.

Id. at .003.

44. RFC 929 includes a “protocol parameter” for selecting the protocol to be offloaded. TCP, UDP and IP were among the protocols to be offloaded:

Generic	Specific	Comment
GIP	IP	Datagram Internetwork Protocol
HHP	TCP	Connection Transport/Host-Host Protocol
GDP	UDP	Datagram Transport/Host-Host Protocol
VTP	TEL	Virtual Terminal (Telnet) Protocol
GFP	FTP	File Transfer Protocol
MAIL	SMTP	Mail Transfer Protocol
PROX	PROX	Proximate Net Interface Protocol

Id. at .013.

3. Mediation Levels

45. The 1984 proposal to standardize offload implementations in RFC 929 is evidence that there was already much activity in offload implementations at that time. The authors of RFC 929 anticipated different types of outboard processors and recognized that the amount of work to be done by the outboard processor might vary from none to partial to full offload. To handle this range, a “mediation level” parameter was proposed.

The mediation level parameter is an indication of the role the Host wishes the OPE to play in the operation of the protocol. The extreme ranges of this mediation would be the case where the Host wished to remain completely uninvolved, and the case where the Host wished to make every possible decision. The specific interpretation of this parameter is dependent upon the particular off-loaded protocol.

The concept of mediation level can best be clarified by means of example. A full inboard implementation of the Telnet protocol places several responsibilities on the Host. These responsibilities include negotiation and provision of protocol options, translation between local and network character codes and formats, and monitoring the well-known socket for incoming connection requests. The mediation level indicates whether these responsibilities are assigned to the Host or to the OPE when the Telnet implementation is outboard. If no OPE mediation is selected, the Host is involved with all negotiation of the Telnet options, and all format conversions. With full OPE mediation, all option negotiation and all format conversions are performed by the OPE. An intermediate level of mediation might have ordinary option negotiation, format conversion, and socket monitoring done in the OPE, while options not known to the OPE are handled by the Host.

The parameter is represented with a single ASCII digit. The value 9 represents full OPE mediation, and the value 0 represents no OPE mediation. Other values may be defined for some protocols (e.g., the intermediate mediation level discussed above for Telnet). The default value for this parameter is 9.

Id. at .015-.016.

46. More than a decade passed between the publication of RFC 929 and the priority date of the earliest Alacritech provisional application. During that time, protocol offload was the subject of many papers and systems across the range anticipated by RFC 929. These implementations can be categorized based on the three principal dimensions of protocol offload: 1) The set of protocols to be offloaded (e.g. TCP/IP, VMTP, OSI), 2) the portions of the protocol that are offloaded (e.g. full offload, partial offload, fast path offload, no offload), 3) the offload implementation (e.g. parallel processor, standard microprocessor, custom processor, custom hardware). The cited references below include many different combinations of these three dimensions, but it should be noted that each cited combination was primarily a design decision among a small, finite number of choices. It would have been obvious to alter these implementations along one or more of the dimensions for a new implementation that would have produced predictable results. In other words, it was well recognized that depending on the application, it was desirable to vary the extent of offloading. The simplest example is that while offloading the entire protocol may seem on the surface advantageous, it was expensive because handling every type of data packet requires a complex offloading device. For example, it was well known that setting up a connection and entering the ESTABLISHED state was much more complex than simply

receiving and sending data packets. Ex.1006, Tanenbaum96 at .583 (“The key to fast TPDU processing is to separate out the normal case (one-way data transfer) and handle it specially. Although a sequence of special TPDUs are needed to get into the *ESTABLISHED* state, once there, TPDU processing is straightforward until one side starts to close the connection.”).

C. Offloaded Protocols

47. By the mid-1990s, TCP/IP was becoming a predominant network standard, but many other networks were still in use and new network protocols were being investigated.

1. OSI Protocol Offload

48. OSI protocol offload engines were built and tested by Thia and Woodside. Ex.1015, Thia and Ex.1038, Woodside.

2. TCP/IP Protocol Offload

49. TCP/IP offload engines were built or described by many in the field including Bach, Erickson, Morris, Cooper, Kung, Rütsche and Chesson. Ex.1020, Bach; Ex.1005, Erickson; Ex.1021, Morris; Ex.1022, Cooper; Ex.1023, Kung; Ex.1017, Rütsche92; Ex.1018, Rütsche93; Ex.1024, Chesson.

3. VMTP and XTP Protocol Offload

50. VMTP and XTP were proposed as alternatives to TCP. A VMTP offload engine was described by Kanakia, and an XTP protocol accelerator was described by Chesson. Ex.1025, Kanakia; Ex.1024, Chesson.

4. Multi-Protocol Offload

51. General-purpose offload engines were also proposed. Erickson discloses a range of protocol scripts for offloading different protocols.

Each type of protocol will have its own script. Types of protocols include, but are not limited to, TCP/IP, UDP/IP, BYNET lightweight datagrams, deliberate shared memory, active message handler, SCSI, and File [sic:Fibre] Channel.

Ex.1005, Erickson at 5:47-51.

52. Kung and Cooper describe the Nectar network-based multicomputer system in which the processors communicate via Communications Acceleration Boards (CABs) that can run different protocols.

The CAB runtime system currently supports several transport protocols with different reliability/overhead tradeoffs [10]. They include the standard TCP/IP protocol suite besides a number of Nectar-specific protocols.

Ex.1026, Kung and Cooper at .003.

D. Portions of the Protocol Offloaded

53. The portion of the protocol offloaded (called “mediation level” in RFC 929) falls into several types that range from partial offload to full offload. That is, either part of the protocol processing can be offloaded (partial offload) or the entire protocol processing can be offload (full offload).

1. Checksum Offload

54. One of the first parts of protocol processing to be offloaded was the checksum calculation (a partial offload). An adapter doing only checksum offload is less complex because it does not require the adapter to maintain the connection state.

55. Dalton describes the HP Afterburner card with optional hardware for checksum calculation:

To support the use of the on-card memory as clusters, we have written a small number of functions. The most important is a special copy routine, functionally equivalent to the BSD function bcopy. It is optimized for moving data over the I/O bus, and also optionally uses the card's built-in unit to calculate the IP checksum of the data it moves. Another function converts a single-copy cluster into a chain of normal clusters and mbufs; it also calculates the checksum.

Ex.1027, Dalton at .011 (emphasis added).

2. Full Offload

56. Exemplary full offload papers and systems include Murphy, Bach, MacLean, Cooper and Rüttsche.⁵ Ex.1028, Murphy; Ex.1020, Bach; Ex.1029, MacLean; Ex.1022, Cooper; Ex.1017, Rüttsche92; Ex.1018, Rüttsche93.

⁵ In a “full offload,” the adapter does not typically initiate connections on its own. The host initiates the connection by opening a socket to an IP address and TCP

3. Multi-Level Offload

57. Chesson describes a protocol chip plus an optional control processor that can do a range of offloads from partial (checksum, sequence numbers, etc.) to full offload. Ex.1024, Chesson.

4. Header Prediction

58. In 1988, Van Jacobson proposed a header prediction algorithm for improving the performance of TCP/IP implementations. This “header prediction” teaching led to various types of partial offload. The code, which uses header templates, is partitioned into one module for the commonly executed path (the fast path) and another module to handle the more complex cases and exception handling (the slow path).

59. Code to implement the header prediction algorithm was incorporated in the BSD 4.4-Lite distribution.

Most IP packets carry no options. Of the 20-byte header, 14 of the bytes will be the same for all IP packets sent by a particular TCP connection. The IP length, ID, and checksum fields (6 bytes total) will probably be different for each packet. Also, if a packet carries any

port. The host establishes the connection and directs the stack of protocol layers to create the connection. Yet those of skill in the art often still refer to such systems as “full offload.”

options, all packets for that TCP connection will be likely to carry the same options.

The Berkeley implementation of UNIX makes some use of this observation, associating with each connection a template of the IP and TCP headers with a few of the fixed fields filled in. To get better performance, we designed an IP layer that created a template with all the constant fields filled in. When TCP wished to send a packet on that connection, it would call IP and pass it the template and the length of the packet. Then IP would block-copy the template into the space for the IP header, fill in the length field, fill in the unique ID field, and calculate the IP header checksum.

This idea can also be used with TCP, as was demonstrated in an earlier, very simple TCP implemented by some of us at MIT [6]. In that TCP, which was designed to support remote login, the entire state of the output side, including the unsent data, was stored as a preformatted output packet. This reduced the cost of sending a packet to a few lines of code.

A more sophisticated example of header prediction involves applying the idea to the input side. In the most recent version of TCP for Berkeley UNIX, one of us (Jacobson) and Mike Karels have added code to precompute what values should be found in the next incoming packet header for the connection. If the packets arrive in order, a few simple comparisons suffice to complete header processing.

Ex.1030, Clark at .003.

60. The 1995 book (Stevens2) walks through the Jacobson BSD header prediction code including the conditions for selecting the fast or slow path. In order to take the fast receive path, six conditions must be met, including:

1. The connection must be ESTABLISHED.
2. The following four control flags must not be on: SYN, FIN, RST, or URG. The ACK flag must be on.
- 3.-6. [Conditions to assure that the received segments are in-order]

Ex.1013, Stevens2 at .962-.963.

a) Partial Offload with Header Prediction

61. The fast and slow paths described by Stevens gave a natural division for protocol offload implementations. Building on the Jacobson BSD header prediction code, Biersack (Ex.1016) describes TCP protocol offload with fast and slow paths. Thia and Woodside (Ex.1015) also build upon the Jacobson BSD header prediction algorithm and apply its teachings to derive an OSI protocol offload with the fast path implemented in hardware.

62. The header prediction code in the FreeBSD release is also discussed in the Alacritech 1997 Provisional application:

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, Alacritech 1997 Provisional Application at .057.

63. Thus, the Jacobson header prediction code forms the basis of what Alacritech offloads to its intelligent network interface card (INIC).

E. Offload Implementation

64. Offloading the transport layer to an interface card was discussed in Tanenbaum96:

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.

Ex.1006, Tanenbaum96 at .498 (emphasis added).

65. Others have disclosed more details of offload hardware including implementations based on multiprocessors, microprocessors, custom processors and custom logic.

1. Multiprocessor Offload

66. Several groups proposed or built systems in which protocol processing is offloaded from the application processor to one or more dedicated processors in a multiprocessor configuration to protocol processing.

67. The Nectar system:

The Nectar communication processor together with its host can be viewed as a (heterogeneous) shared-memory multiprocessor. Dedicating one processor of a multiprocessor host to communication tasks can achieve some of the benefits of the Nectar approach, but this

constrains the choice of host operating system and hardware. In contrast, the Nectar communication processor has been used with a variety of hosts and host operating systems.

Ex.1022, Cooper at .006.

68. The Parallel Protocol Engine:

In this paper our goal is to demonstrate that a careful implementation of a standard transport protocol stack on a general-purpose multiprocessor architecture allows efficient use of the bandwidth available in today’s high-speed networks. As an example, we chose to implement the TCP/IP protocol suite on our 4-processor prototype of the PPE.

Ex.1017, Rütsche92 at .009.

69. Rütsche also designed a Gb/s Multimedia Protocol Adapter based on the PPE:

In this paper we present a new multiprocessor communication subsystem architecture, the Multimedia Protocol Adapter (MPA), which is based on the experience with the Parallel Protocol Engine (PPE) [Kaiserswerth 92] and is designed to connect to a 622 Mb/s ATM network. The MPA architecture exploits the inherent parallelism between the transmitter and receiver parts of a protocol and provides support for the handling of new multimedia protocols.

Ex.1018, Rütsche93 at .001.

2. Offload Adapters based on Microprocessors

70. Protocol offloading may be implemented by executing code in one or more microprocessors on an intelligent network interface card or on a network accelerator board used in conjunction with a standard NIC (network interface card).

71. Kanakia describes a network adapter board with a microprocessor and other support chips:

The prototype Network Adapter Board (NAB) has been designed using Motorola’s MC68020 as the on-board processor, running at 16 Mhz clock rate; it uses about 200 hundred standard MSI and LSI components. The current version is designed for connecting two VMP multiprocessor system with a 100 megabit/sec point-to-point connection.

Ex.1025, Kanakia at .010.

72. MacLean describes microprocessor-based protocol accelerators residing on a VME card:

The internal functions and data flows of the protocol accelerator shown in Figure 2. We use a dual CPU approach to protocol processing, with one CPU subsystem dedicated to the transmission, and the other to the reception. The transmit and receive CPUs are both 68020 (25 MHz) based, each with its own private resources: ROM,

parallel I/O, interrupt circuitry and 128 kilobytes of random access memory (RAM). In addition there is 128 kilobytes of RAM shared by both CPUs which is also accessible to the two host busses, VME and VSB.

Ex.1029, MacLean at .004.

73. Rüttsche describes a multimedia protocol adapter (MPA) using a pair of “transputer” microprocessors:

The selection of the inmos² T9000 [inmos 91] is based on our good experience with the transputer family of processors in the PPE. The most significant improvements of the T9000 over the T425 for protocol processing are faster programmable link interfaces, a faster memory interface, and a cache.

Ex.1018, Rüttsche93 at .003.

3. Offload Adapters based on Custom Processors or Custom Logic

74. Other designers have proposed custom processors and/or custom logic for protocol offload. Chesson describes a Protocol Engine chipset for real-time protocol processing. Depending on the amount of protocol offload desired, an adapter can be built with or without the custom control processor (CP):

The Protocol Engine® chipset offers real-time protocol processing for high-speed networks. A wide range of cost-performance subsystem solutions are available through various configurations based on the PE Chipset. The chipset (shown in Figure 1) consists of four chips:

MPORT, HPORT, BCTL, and CP. A basic configuration consists of MPORT, HPORT, and BCTL.

Ex.1024, Chesson at .006.

75. The optional Chesson Control processor is a custom processor designed for fast protocol processing:

Control Processor (CP) of the Protocol Engine® chipset is a 32-bit, multi-thread execution unit that provides high speed protocol processing.

Id. at .039.

76. Thia also discloses the design of a custom VLSI chip for protocol offload:

The chip design based on bypassing is called ROPE, for Reduced Operation Protocol Engine. The contribution of this paper is to define the host/chip interface and the chip operation, and to report on a VHDL-based feasibility study of the chip design. It appears to be feasible to support an end-system single-connection data rate approaching 1 Gbps.

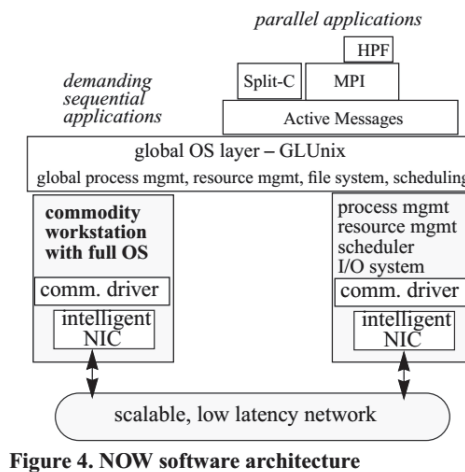
Ex.1015, Thia at .002.

77. Culler describes the Berkeley Network of Workstations (NOW) in which the Active Messages protocol is offloaded to intelligent NICs built with Myricom LANai chips:

The hardware configuration of the Berkeley NOW system consists of one hundred and five Sun Ultra 170 workstations, connected by a

large Myricom network[Bode95], and packaged into 19-inch racks. Each workstation contains a 167 MHz Ultra1 microprocessor with 512 KB level-2 cache, 128 MB of memory, two 2.3 GB disks, ethernet, and a Myricom “Lanai” network interface card (NIC) on the SBus. The NIC has a 37.5 MHz embedded processor and three DMA engines, which compete for bandwidth to 256 KB of embedded SRAM. The node architecture is shown in Figure 1.

Ex.1032, Culler at .001.



Id. at .003.

78. Alteon describes their third generation intelligent Ethernet adapter that includes performance improvements from protocol offload, reduction in memory copies and reduction of interrupts.

Using an intelligent adapter with an onboard RISC-based processor specially designed for embedded application processing, Alteon’s Gigabit Ethernet technology not only reduces the number of times

data is copied among processing entities, it allows a single interrupt to be issued for multiple data packets—radically altering the ratio of interrupts to packets, and eliminating the scalability problems inherent in older adapter designs.

Ex.1033, Alteon at .022.

79. HP discloses a custom chip called Tachyon that includes send offload, receive offload, hardware checksum calculation, DMA, and headers/data splitting:

To provide support for customer networking applications, Tachyon:

- Manages the protocol for sending and receiving network sequences over Fibre Channel.
- Provides complete support of networking connections.
- Computes exact checksums for outbound IP packets and inserts them in the data stream, thereby offloading the host of a very compute-intensive task.
- Computes an approximate checksum for inbound IP packets that partially offloads the checksum task from the host.
- Contains hardware header/data splitting for inbound SNAP/IP sequences.

Ex.1034, Smith at .004.

F. Protocol Offload Summary

80. The preceding paragraphs have shown many offload implementations foreshadowed by RFC 929 described above. These implementations include many variations along the three dimensions of network protocol offload: 1) the set of protocols to be offloaded, 2) the portions of the protocol that are offloaded, and 3) the offload implementation. The citations show that each of the individual concepts was well known and that many different combinations along the three dimensions were successfully implemented by practitioners. It would have been obvious to alter these implementations along one or more of the dimensions for a new implementation that would have produced predictable results.

G. Additional Background Technology

81. Protocol offload adapters have incorporated many well-known design techniques originally developed for general purpose processors. Some of these concepts, such as DMA and virtual memory, are briefly described below. More information is available from textbooks on Computer Architecture. *See e.g.*, David A. Patterson and John L. Hennessy, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Publishers Inc., San Mateo, CA, USA., 1990. (Ex.1035, Patterson).

1. DMA

82. DMA (Direct Memory Access) is a hardware-based technique for transferring data between memory systems or between a host memory and an I/O device.

Since I/O events so often involve block transfers, direct memory access (DMA) hardware is added to many computer systems to allow transfers of numbers of words without intervention by the CPU.

Ex.1035, Patterson at .566. Hardware that processes the received TCP/IP packets on the fast path must make the received data accessible to the application running on the host machine (e.g., in the designated host memory as discussed above).

DMA was a common and efficient way to achieve this, as I discuss below.

83. Before DMA was common, processors used I/O (input/output) instructions to transfer data to I/O devices. A benefit of using DMA is that fewer

processor cycles are required to transfer the data. With DMA, the DMA engine is loaded with an address and count of data to be moved, then the data movement proceeds while the processor is doing other tasks. In some implementations, DMA engines are under the control of a host processor, while in others a DMA engine is controlled by an intelligent controller on an I/O adapter. The DMA engine itself may be located either in the host or on an I/O adapter.

84. DMA may be used either to read from host memory or to write to host memory. In some implementation, there are separate send and receive DMA engines and in others, a common DMA engine can be programmed to transfer to or from host memory:

Outbound Block Mover. The outbound block mover block’s function is to transfer outbound data from host memory to the outbound sequence manager via DMA. It takes as input an address/length pair from the outbound sequence manager block, initiates the Tachyon system interface bus ownerships, and performs the most efficient number and size of transactions on the Tachyon system interface bus to pull in the data requested.

...

Inbound Block Mover. The inbound block mover is responsible for DMA transfers of inbound data into buffers specified by the multiframe sequence buffer queue, the single-frame sequence buffer queue, the inbound message queue, or the SCSI buffer manager. The

inbound block mover accepts an address from the inbound data manager, then accepts the subsequent data stream and places the data into the location specified by the address.

Ex.1034, Smith at .007, .009.

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.

Ex.1015, Thia at .007.

Bus Controller (BC): The BC is a programmable busmaster DMA controller. It provides a small FIFO and a table for DMA requests. The FIFO contains a pointer to the linked list of source data and a connection identifier. The BC determines the destination memory address through the connection identifier in the table. The list format is the same for the BC and the DMAU. In the transmit BC the host writes to the FIFO and the protocol processor to the table. In the receive BC the protocol processor writes to the FIFO and the host to the table.

Ex.1018, Rütsche93 at .004-.005.

2. Virtual and Physical Memory Addresses

85. I/O adapters that transfer data directly to or from memory need to be provided with the memory addresses of the buffers. Many processors use virtual addressing in which large buffers appear to the processor as single contiguous memory space even though the addressed pages may not be contiguous in physical

memory. To translate from virtual to physical memory addresses, the processor uses page tables that store the appropriate mappings from virtual to physical pages.

With virtual memory, the CPU produces *virtual addresses* that are translated by a combination of hardware and software to *physical addresses*, which can be used to access main memory. This process is called *memory mapping* or *address translation*.

Ex.1035, Patterson at .465 (emphasis in original).

86. In order for an I/O device to access the main memory buffers, either the physical address may be supplied for each page, or a translation table may be maintained on the I/O controller to allow it to operate on virtual addresses. Erickson has a “physical address buffer map” in the adapter memory and discusses some options for handling the translation:

The vtophys() function performs a translation of the user-provided virtual address into a physical address usable by the adapter. In all likelihood, the adapter would have a very limited knowledge of the user process’ virtual address space, probably only knowing how to map virtual-to-physical for a very limited range, maybe as small as a single page. Pages in the user process’ virtual address space for such buffers would need to be fixed. The udpscript procedure would need to be enhanced if the user data were allowed to span page boundaries.

Ex.1005, Erickson at 8:14-24.

87. Tanenbaum96 suggests the use of direct copying into the user buffer to avoid unnecessary copy. Ex.1006, Tanenbaum96 at .585. Tanenbaum96 also

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.1006, Tanenbaum96 at .579, .582.

88. In implementations that avoid the extra copy steps, the DMA engine transfers the reassembled original byte stream into user space for use by the application layer. Once the headers are processed, the application has no need for those headers, as demonstrated by the illustration of the protocol stack above in Section V.B.1. *See* Ex.1006, Tanenbaum96 at .055-56, .541. For example, Tanenbaum96 discloses that the headers are checksummed before transferring data to user space to verify that it is going to the correct location:

“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.”

Id. at .589. The TCP/IP headers are added (and stripped off) by intermediate layers as the data moves through the protocol stack.

VI. OVERVIEW OF 948 PATENT

89. The 948 Patent relates to offloading TCP protocol processing for established TCP connections to a network interface card (NIC). Ex.1001, 948

Patent at Abstract. The specification of the 948 Patent refers to the disclosed NIC, which performs offloading, as an “intelligent network interface card (INIC).” *See id.* at Abstract.

90. The INIC of the 948 Patent permits two modes of operation: a “fast path” in which protocol processing from the physical layer through the TCP layer bypasses the host protocol stack and is instead performed on the INIC, and a “slow path” in which network frames are handed to the host at the MAC layer and passed up through the host protocol stack conventionally. The concept is illustrated in Fig. 6, shown below:

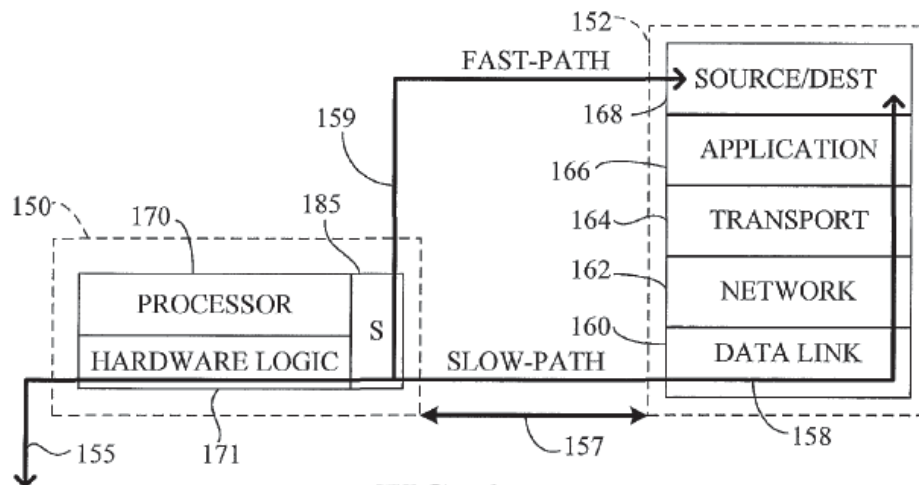


FIG. 6

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, 948 Patent at Fig.6; 9:11-29.

91. When a connection is created, the host computer creates a connection record that is referred to in the 948 Patent as a “Communication Control Block (CCB).” Ex.1001, 948 Patent at 5:22-29. This contains similar information as in the Transmission Control Block (TCB) in RFC791, namely connection and state information for the connection. Ex.1001, 948 Patent at 12:32-39. When the INIC receives a packet, it checks whether a connection exists by looking for a CCB corresponding to the connection information in the received packet header. The INIC uses the result of this comparison to determine which “path” should be used for a received packet. If the connection exists, the packet is processed 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.

Ex.1001, 948 Patent at 11:57-63; 12:6-10.

92. The claims of the 948 Patent are directed to fast-path TCP receive processing when a connection is in the ESTABLISHED state (in other words, there are no exceptions such as no IP fragmentation, or one of the SYN, FIN or RST flags set in the received packet).

VII. 948 PATENT PROSECUTION HISTORY

93. I have reviewed the prosecution history of the 948 Patent. There were no rejections or amendments during the prosecution of the 948 Patent.

94. 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.

95. 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; 948 File History at .075-.99. Neither Thia nor Tanenbaum⁹⁶ were discussed during the prosecution of the application leading to the 948 Patent.

96. On June 20, 2014, the Examiner gave the following reasons for allowance:

2. The following is an examiner’s statement of reasons for allowance: 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 a 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, 948 File History at .117.

VIII. CLAIM CONSTRUCTIONS

A. Legal Standard

97. I understand that in deciding whether to institute *inter partes* review, “[a] claim in an unexpired patent shall be given its broadest reasonable construction in light of the specification of the patent in which it appears.” 37 C.F.R. § 42.100(b). I further understand that “the broader standard serves to

identify ambiguities in the claims that can then be clarified through claim amendments.” Final Rule, 77 Fed. Reg. 48680, 48699 (Aug. 14, 2012).

98. In forming my opinions as set forth in this declaration, I have accorded all claim terms in claims 1, 3, 6-9, 11, 14-17, 19, and 21-22 in the 948 Patent their broadest reasonable interpretation, as would be understood by a person of ordinary skill in the art at the time of the alleged invention of the alleged invention of the 948 Patent.

IX. THE PRIOR ART

A. Thia: Thia, A Reduced Operation Protocol Engine (ROPE) for a multiple-layer bypass architecture (1995)⁶

99. Thia describes a hardware protocol engine for fast-path data transfer, where the hardware bypasses the host protocol stack for certain packets:

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, Thia at .001. Thia is based on the Open System Interconnect (OSI) protocol that I discussed above in Section V.A.1.

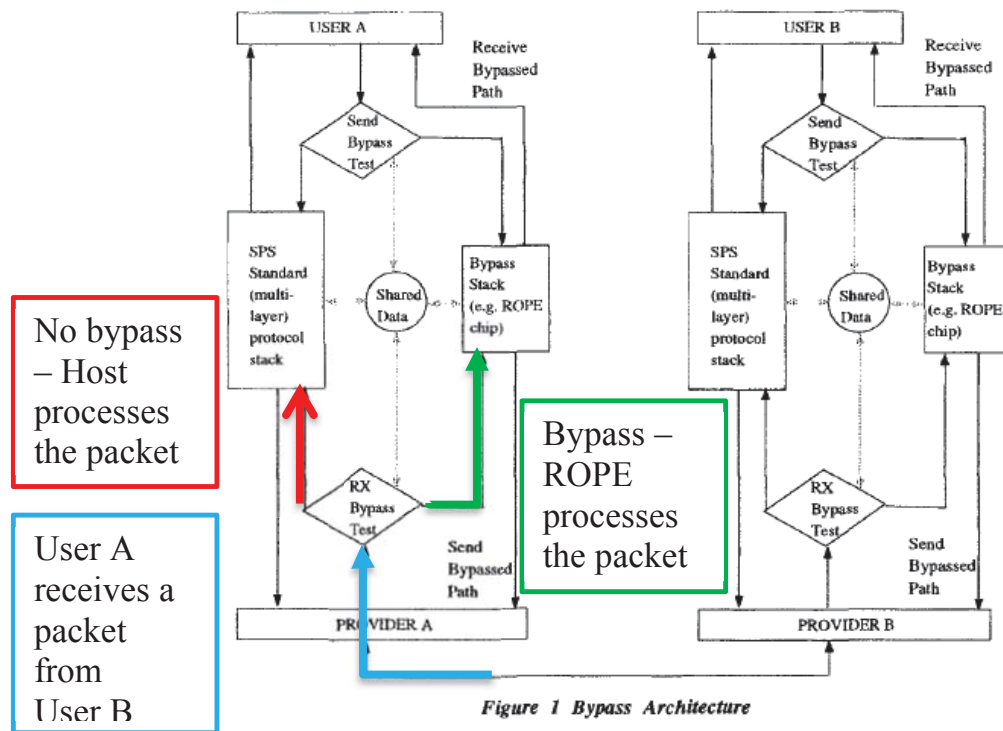
⁶ Thia was published in 1995. I understand that it is prior art because it was published before October 14, 1997, the date to which Alacritech claims priority. See Ex.1015, Thia.

100. Thia’s hardware protocol offload system compares the incoming packet headers with a template that identifies “predicted bypassable headers” (in other words, those that are in a “data transfer phase” (consecutive packets for the same connection). Thia refers to the ROPE hardware as a reduced operation protocol engine because it only handles this subset of packets. Ex.1015, Thia at .004 (“The number of possible PDU formats in the bypass path is reduced to data transfer PDUs”). Thia discloses the use of a receive bypass test (RX bypass test in Figure 1) where the ROPE hardware performs all protocol processing for packets in this “data transfer phase,” bypassing the standard protocol stack (SPS) (host protocol stack):

2.1 Bypass Architecture

Figure 1 illustrates the architecture of a bypass implementation for any standard protocol. The standard protocol stack (SPS) is the processing path taken by all PDUs [Protocol Data Units i.e. packets] during a connection without the bypass.... 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. The shared data are used to maintain state consistency between the SPS and the bypass stack, including window flow control parameters and connection identifiers.

Ex.1015, Thia at .003. This “receive bypass test” is performed on the ROPE (i.e., the Network Interface Adapter). Ex.1015, Thia at .006. I’ve illustrated the fast and slow path processing disclosed by Thia below.



Id. at .003 (annotated).

101. 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 Stephens2 references discussed below. *Id.* at .002. Thia teaches that the receive bypass test ensures that the protocol bypass on the ROPE is for packets without exceptions – i.e., those in the “data transfer phase” (this is also

what Stevens² referred to as the ESTABLISHED state for TCP) and that the SPS on the host handles packets in the 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.

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., .004.

102. The receive bypass test “matches” the incoming PDU (packet) headers with a template that identifies the next in sequence (predicted) bypassable headers. *Id.* at .003. To be part of the data transfer, the received packets must indicate that they are for the same connection. A POSA would have understood that the headers would be parsed to identify the fields to be matched with the

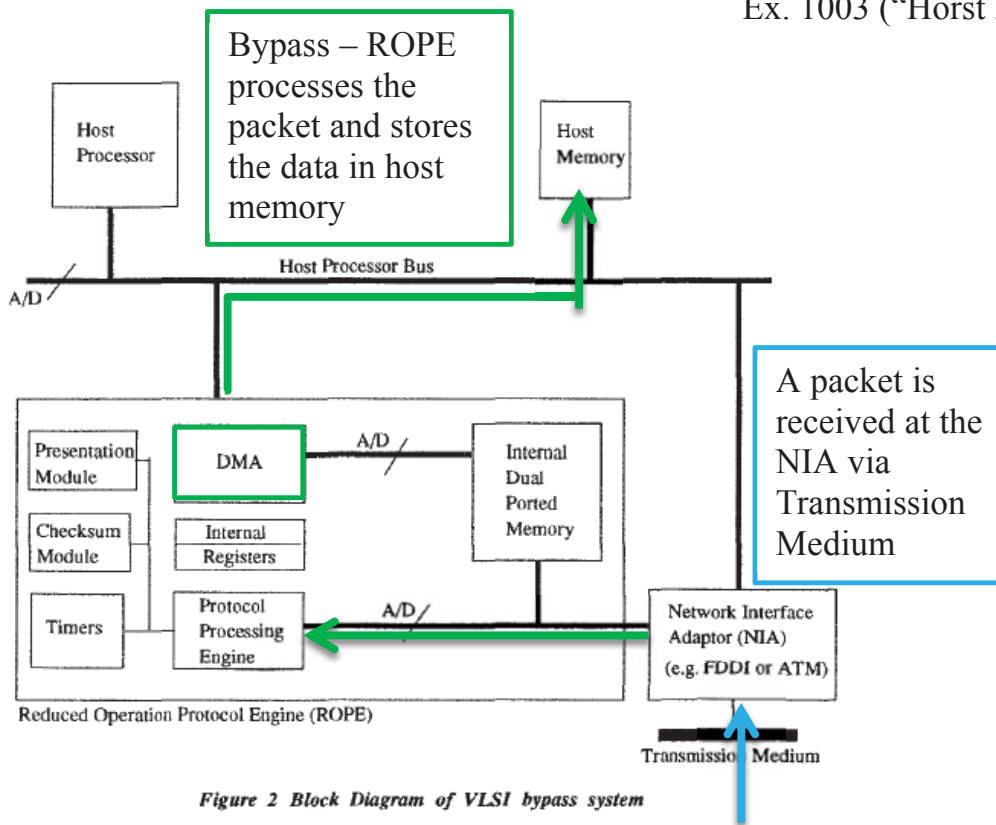
template. The bypass stack then “performs all the relevant protocol processing in the data transfer phase.” *Id.* As I described above in Section V.A, on the receive side, this includes stripping off the headers and handing up the payload to the layer above. The bypass stack on the ROPE performs header decoding and can perform the checksum on the transport layer. *Id.* at .006.

103. Thia’s DMA then copies the data to Host Memory, eliminating data copying within layers. *Id.* at .007 (“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.”) (emphasis added). A POSA would understand that only the payload data is transferred. The headers need not be transferred because they are decoded and checked by the bypass chip. Highlighted Table 1 and Figure 2 of Thia show these functions:

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

Id. at .006 (annotated). In other words, a POSA would have understood that the transport header is stripped off before the data is transferred to the layer above as described in Section V.B.1



Id. at .007 (annotated).

104. This indicates that creating the bypass for certain operations was implemented with minimal changes to the original software. *Id.* at .002. This teaches that using a bypass for multiple layers (i.e., concatenation of processing procedures in adjacent layers in the data transfer phase) provides additional gains, including avoiding the overhead of encoding and decoding the interface control information passed between layers and the queueing of data at layer boundaries. *Id.* at .004.

B. Tanenbaum96: A. Tanenbaum, Computer Networks, 3rd ed. (1996)⁷

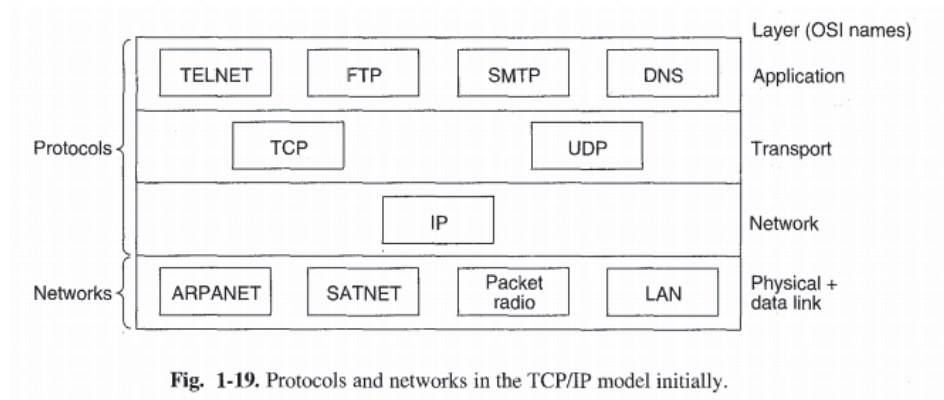
105. Tanenbaum96, “Computer Networks,” is a 700+ page text book covering network hardware, software, protocols and standards. It is a third edition of the 1981 Tanenbaum book. Tanenbaum96 is a widely-cited textbook covering network hardware, software, protocols (including OSI and TCP/IP) and standards. The 1996 edition is cited and incorporated by reference in the 948 Patent.

106. Tanenbaum96 describes both TCP and UDP protocols. Note that UDP, unlike TCP, is connectionless and thus does not require setting up a connection:

The Internet has two main protocols in the transport layer, a connection oriented protocol and a connectionless one. In the following sections we will study both of them. The connection-oriented protocol is TCP. The connectionless protocol is UDP.

Ex.1006, Tandenbaum96 at .539.

⁷ Tanenbaum96 was a well-known resource to a POSA. I understand that it is prior art because it was published before October 14, 1997, the date to which Alacritech claims priority. *See* Ex.1006, Tanenbaum96.



Id. at .055, Fig 1-19.

107. Tanenbaum96 recognizes that an “obstacle to fast networking is protocol software,” and teaches “fast path” processing for TCP as a solution. Ex. 1006 at .583-.585. This “fast path” solution is based off “header prediction.” See Section V.E.4 above for a description of “header prediction.”

108. Tanenbaum96 teaches that header prediction is based on the principle that fast path processing should apply to normal data transfers:

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

Ex.1006, Tanenbaum96 at .583.

109. Tanenbaum96 teaches fast path transmissions using a prototype header stored in the transport entity, because in the normal case of an established TCP connection, only a few fields of the header change in consecutive packets.

Compare Section V.B.5.a. (describing complexity of opening a connection, i.e., a “socket”). In other words, the transport entity only needs to change a few fields to send subsequent packets:

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

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

Id. at .583 (emphasis added).

110. The fast path send and receive processing are illustrated in Figure 6-49 of Tanenbaum96.

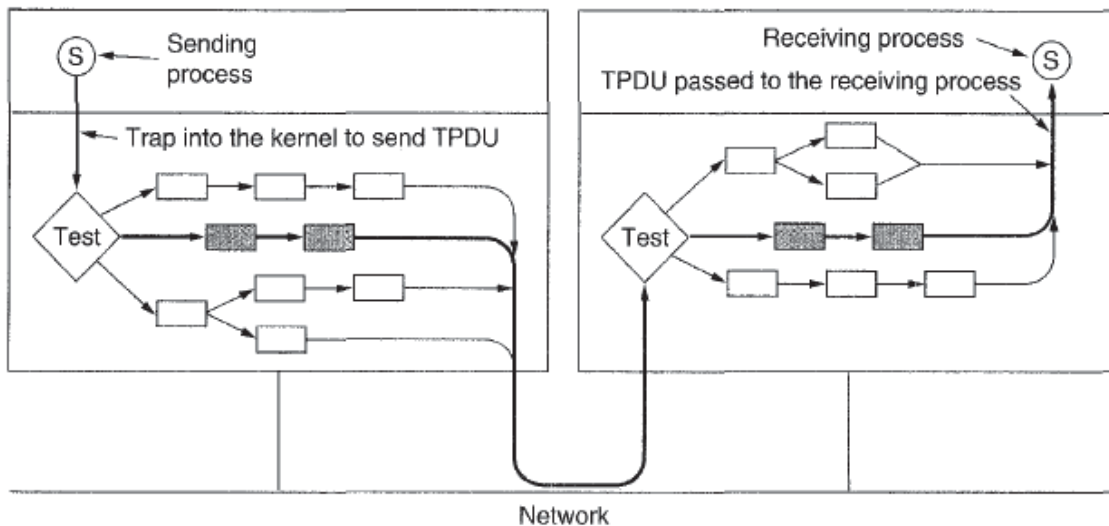


Fig. 6-49. The fast path from sender to receiver is shown with a heavy line. The processing steps on this path are shaded.

Ex.1006, Tanenbaum96 at .584.

111. Tanenbaum96 teaches that the transport entity can be implemented by the host operating system, or can be offloaded to the NIC (e.g., as a processor on the NIC):

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).

112. Tanenbaum96 discloses that the TCP transport entity divides data streams into TCP segments for subsequent transmission. *See* Section V.B.8.

(segmentation description). The receiving TCP transport entity reconstructs the byte stream from the received TCP segments.

Each machine supporting TCP has a TCP transport entity, either a user process or part of the kernel that manages TCP streams and interfaces to the IP layer. A TCP entity accepts user data streams from local processes, breaks them up into pieces not exceeding 64K bytes (in practice, usually about 1500 bytes), 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.

Id. at .540. A POSA would understand that the byte stream is transferred and stored into application memory without the headers from the layers below.

113. Tanenbaum96 goes on to describe a TCP prototype header (i.e., a header template that is used to create additional headers for sending packets) and offloading protocol processing by the transport entity in detail:

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

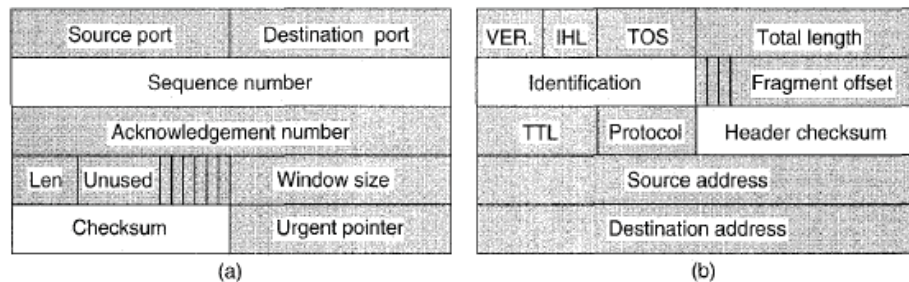


Fig. 6-50. (a) TCP header. (b) IP header. In both cases, the shaded fields are taken from the prototype without change.

Id. at .584 (emphasis added).

114. Tanenbaum96 also teaches TCP fast path receiving by looking up a TCP connection record based on the IP source address, TCP source port, IP destination address and TCP destination address, checking to see if it the packet is a normal one in the ESTABLISHED state, and then putting the data into user memory. In other words, Tanenbaum96 is teaching that the transport entity performs this check to determine whether the packet is suitable for fast path processing. See Section V.E.4. (header prediction offload). Note that there may be multiple connections on a single computer, and thus when a packet comes in, it must be checked against the connection records that may represent multiple connections:

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....

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

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

Ex.1006, Tanenbaum96 at .584-.585 (underlining added, bold in original). Tanenbaum discloses that part of Header Prediction is checking whether the received matches a connection record (i.e., whether the source and destination addresses and ports match). The phrase “the TPDU is a full one” means that it is a full TPDU, in other words, not fragmented. The phrase “the sequence number is

the one expected” means that it is the next packet in sequence as determined by the sequence number, in other words it is not out of order or a retransmission.

115. Tanenbaum discloses the function of the RST, SYN and FIN control flags. When the SYN flag is set, it indicates that the connection is not yet in the ESTABLISHED state. If either the RST or FIN flag is set, that indicates that the connection will be closed (i.e., is to be taken out of the ESTABLISHED state):

The RST bit is used to reset a connection that has become confused due to a host crash or some other reason. It is also used to reject an invalid segment or refuse an attempt to open a connection. In general, if you get a segment with the RST bit on, you have a problem on your hands.

The SYN bit is used to establish connections. The connection request has SYN= 1 and ACK= 0 to indicate that the piggyback acknowledgement field is not in use. The connection reply does bear an acknowledgement, so it has SYN = 1 and ACK = 1. In essence the SYN bit is used to denote CONNECTION REQUEST and CONNECTION ACCEPTED, with the ACK bit used to distinguish between those two possibilities.

The FIN bit is used to release a connection. It specifies that the sender has no more data to transmit. However, after closing a connection, a process may continue to receive data indefinitely. Both SYN and FIN segments have sequence numbers and are thus guaranteed to be processed in the correct order.

Id. at .545.

116. 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:

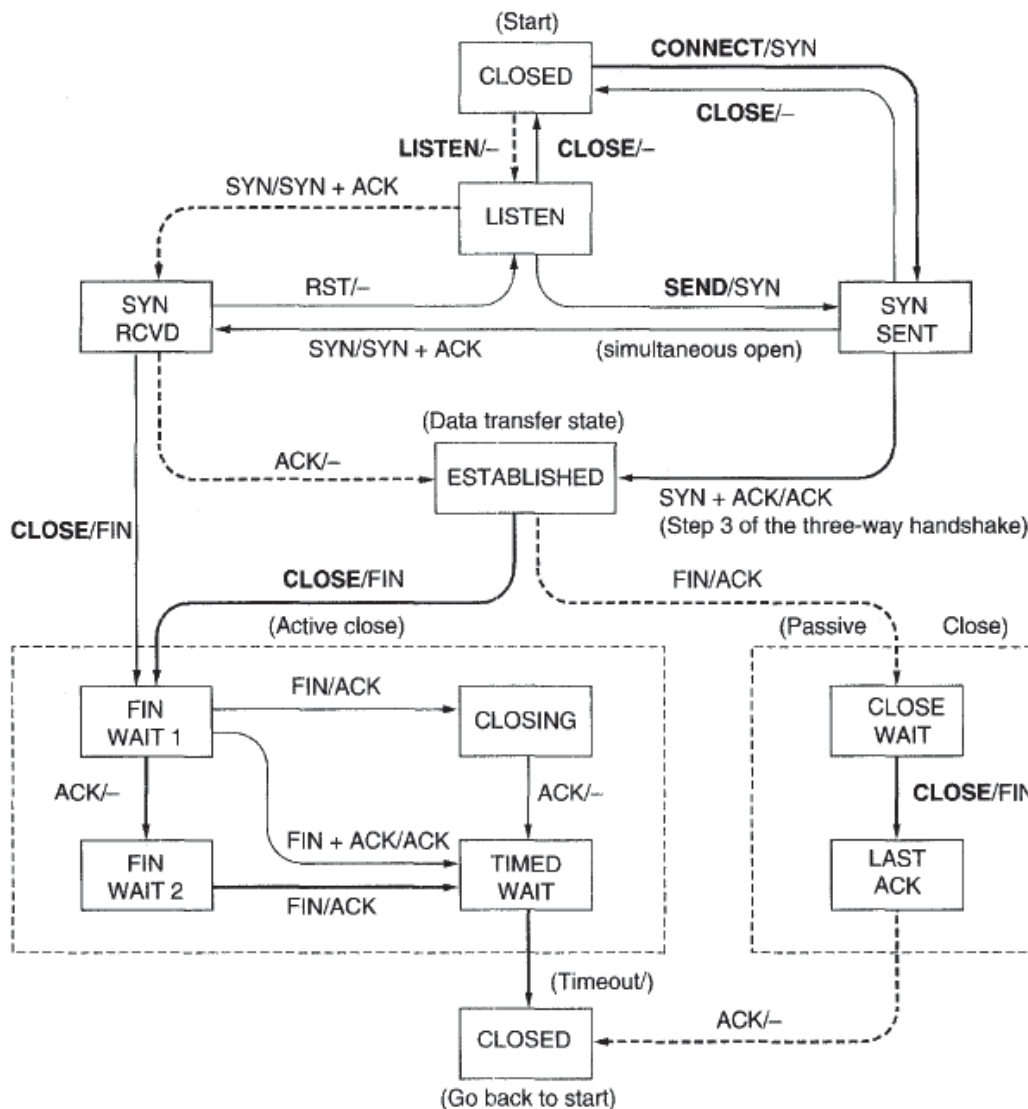


Fig. 6-28. TCP connection management finite state machine. The heavy solid line is the normal path for a client. The heavy dashed line is the normal path for a server. The light lines are unusual events.

Id. at .550. As can be seen from the state diagram, the SYN, FIN and RST control flags 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, Tanenbaum96 at .545, Fig.6-24. Figure 6-24 of Tanenbaum96 illustrates these flags in the TCP header.

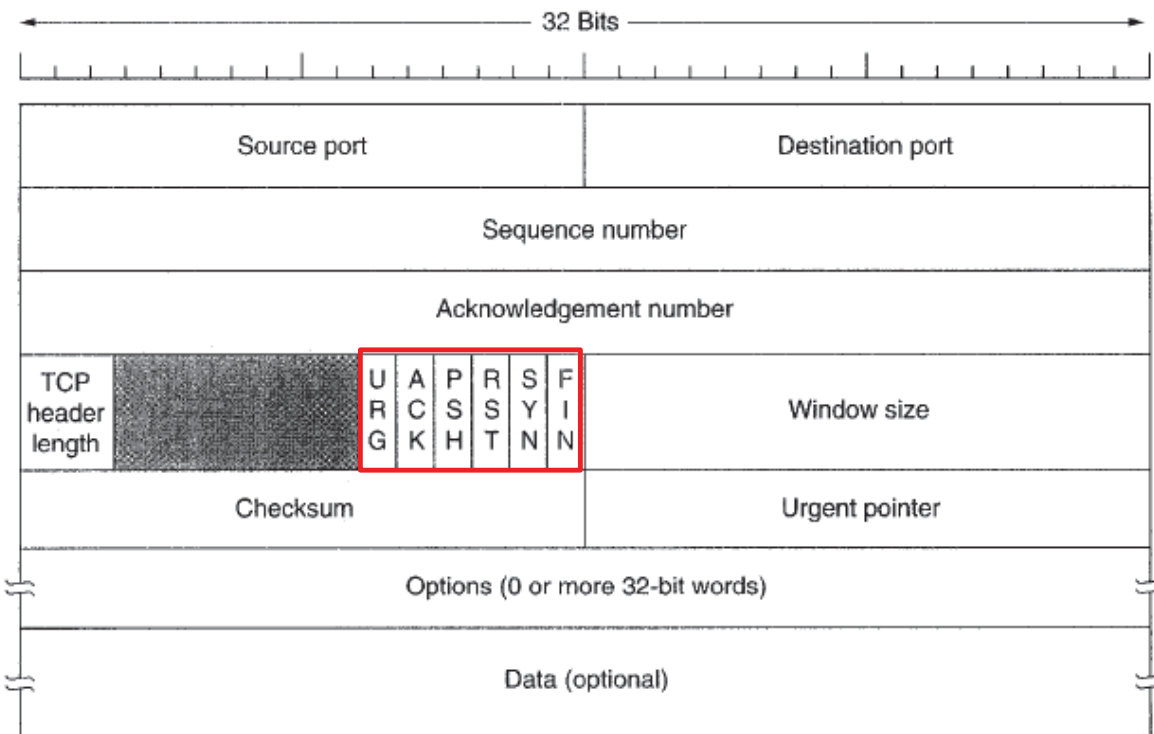


Fig. 6-24. The TCP header.

Id. at .544.

117. The “connection record” disclosed in Tanenbaum96 is used to maintain TCP state:

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

Id. at .549 (emphasis added).

118. The “connection record” is the same as the “Transmission Control Block (TCB)” described in RFC 793, the TCP protocol specification:

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

Ex.1007, RFC 793 at .024 (emphasis added).

119. I describe a TCB and RFC 793 in Section V.B.5.

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

121. Again, there may be multiple connections, and Tanenbaum96 is teaching a technique to quickly lookup the connection record that corresponds to the received packet.

122. Tanenbaum96 discloses the benefits of fast path processing where unnecessary interlayer data copying is limited before the data 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.”).

C. Stevens2: Stevens, TCP-IP Illustrated, Vol. 2

123. Stevens2 is one of the most widely-read and referenced books on the implementation of TCP/IP. Stevens2 is a guide to the source code for TCO/IP implementation in the 4.4BSD-Lite distribution, which was widely-used to implement TCP/IP by companies designing networking products. Stevens2 includes a discussion of BSD’s implementation of Jacobson’s Header Prediction, which was written by Jacobson. Ex.1013, Stevens2 at .960 (“*Header Prediction* was put into the 4.3BSD Reno release by Van Jacobson”). The 948 Patent claims priority to a 1997 provisional, which indicates that “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, 1997 Provisional at .057.

124. Header prediction was described in Stevens2 as follows:

Header prediction helps unidirectional data transfer by handling the two common cases. . . If TCP is 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.

Ex.1031, Stevens2 at .961-.962.

125. Stevens2 includes code and the general algorithm describing Header Prediction that allows for fast path processing in the common case where 1) the connection state is ESTABLISHED, 2) there are no control flags set (SYN, FIN, RST, URG, ACK), and 3) the packet is in-sequence (not out of order or a retransmission).

```
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      /*
369      * If last ACK falls within this segment's sequence numbers,
370      * record the timestamp.
371      */
372      if (ts_present && SEQ_LEQ(ti->ti_seq, tp->last_ack_sent) &&
373          SEQ_LT(tp->last_ack_sent, ti->ti_seq + ti->ti_len)) {
374          tp->ts_recent_age = tcp_now;
375          tp->ts_recent = ts_val;
376      }
377  }
```

Figure 28.11 tcp_input function: header prediction, first part.

Check if segment is the next expected

347-366 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 state must be ESTABLISHED.
2. The following four control flags must not be on: SYN, FIN, RST, or URG. The ACK flag must be on. In other words, of the six TCP control flags, the ACK flag must be set, the four just listed must be cleared, and it doesn't matter whether

PSH is set or cleared. (Normally in the ESTABLISHED state the ACK flag is always on unless the RST flag is on.)

3. If the segment contains a timestamp option, the timestamp value from the other end (*ts_val*) must be greater than or equal to the previous timestamp received for this connection (*ts_recent*). This is basically the PAWS test, which we describe in detail in Section 28.7. If *ts_val* is less than *ts_recent*, this segment is out of order because it was sent before the most previous segment received on this connection. Since the other end always sends its timestamp clock (the global variable *tcp_now* in Net/3) as its timestamp value, the received timestamps of in-order segments always form a monotonic increasing sequence.

The timestamp need not increase with every in-order segment. Indeed, on a Net/3 system that increments the timestamp clock (*tcp_now*) every 500 ms, multiple segments are often sent on a connection before that clock is incremented. Think of the timestamp and sequence number as forming a 64-bit value, with the sequence number in the low-order 32 bits and the timestamp in the high-order 32 bits. This 64-bit value always increases by at least 1 for every in-order segment (taking into account the modulo arithmetic).

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 either a retransmission or a segment beyond the one expected.
5. The window advertised by the segment (*tiwin*) must be nonzero, and must equal the current send window (*snd_wnd*). This means the window has not changed.
6. The next sequence number to send (*snd_nxt*) must equal the highest sequence number sent (*snd_max*). This means the last segment sent by TCP was not a retransmission.

Ex.1013 at .962-.963.

126. The code and algorithm above are implemented at the TCP layer, and thus, do not check for IP fragmentation because the IP layer reassembles fragmented IP packets. Fragmentation, like out of order packets, would require special handling that is not the common case.

X. OBVIOUSNESS COMBINATIONS – MOTIVATIONS TO COMBINE

A. Thia in Combination with Tanenbaum96

127. Thia discloses a chip for fast path receive protocol processing and a bypass test to offload protocol processing of consecutive packets for the same connection in the data transfer phase for the OSI Session and Transport layer protocols. Ex.1015, Thia at .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.

128. In 1996, the Internet and World Wide Web, using TCP/IP, was growing extremely popular. *See generally* Section V.A.-B. Thia looked to header prediction algorithm for TCP/IP. Ex.1015, Thia at .001. Given this, a POSA at this time would have been motivated to implement the TCP/IP fast path protocol processing with the Jacobson header prediction algorithm, using Thia’s Reduced Operation Protocol Engine hardware. 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, 948 Patent at 3:4-8. At the time, there were a finite number of networking protocols, particularly that were as popular as TCP/IP, and thus it would have further been obvious to try to implement TCP/IP using Thia’s ROPE chip on the network interface adapter. *See generally* Section V.A.-B. 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, Tanenbaum96 at .054. Tanenbaum96 also discloses that TCP/IP became the dominant protocol suite as the OSI model vanished. Ex.1006, Tanenbaum96 at .016. A POSA would have been motivated to migrate to the more dominant TCP/IP protocol suite as OSI models vanished. Therefore, Thia’s SPS would include an IP layer in place of the OSI Network layer and a TCP layer in place of the OSI Transport layer with TCP payload data delivered to the Application layer (including functionality from the OSI Presentation and Session layers) running above the transport layer. Thia’s modified bypass test would match the incoming TCP/IP headers with a template such as the one disclosed in Tanenbaum96 for header prediction and, after the bypass stack processing, would copy only the received data to the Host Memory in order without the TCP headers.

129. As I have described in Section V.A.2. and V.B., a POSA would have understood TCP/IP well and standards for TCP/IP are set forth in well-known Request for Comments (RFCs). Accordingly, a POSA would have had a high expectation of success in implementing TCP/IP on Thia’s ROPE chip on the network interface adapter. Tanenbaum96 discloses that the transport processing can be offloaded to a Network Interface Card (e.g., as a processor on the NIC). Tanenbaum96 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.1006, Tanenbaum96 at .579, .582. Thia similarly teaches that “data Copying within layers is eliminated” and discloses the use of a DMA to move data to the Host Memory supporting the goal of “high-speed bulk data transfer.” Ex.1015, Thia at Table 1, .002, .006, .007. Thus, Thia’s fast path processing and DMA would support Tanenbaum’s goals.

130. Thia, like Tanenbaum, discloses a fast path protocol bypass that is based on Jacobson’s Header Prediction Algorithm for TCP/IP. Ex.1015, Thia at .002. Thia states that it’s “receive bypass test” matches incoming headers with a template that identifies the predicted bypassable headers. Thia, like Tanenbaum96, discloses that fast-path processing is to be limited to the normal data transfer state (“OPEN” state) where consecutive packet are received for the same connection in

order to perform a reduced number of protocol operations on the incoming data. Ex.1015, Thia at .004. 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. Ex.1006, Tanenbaum 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 [not out of order].” *Id.* at .583.

131. Combining Tanenbaum96’s TCP/IP and header prediction with Thia would have been understood as combining known methods to yield predictable results. For example, TCP/IP was well known. *See* Section V.B. Header prediction was well known. *See* Section V.E.4. Offloading protocol processing was also generally well known. *See* Section V.B.C.-G.

132. A POSA reading Thia in view of 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 discussed in Sections IX.A and IX.B, 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. *See* Ex.1006, Tanenbaum96 at .584-.585; Ex.1015, Thia at .002. Tanenbaum96 teaches that transport processing can be offloaded to a Network Interface Card, such as the ROPE chip in Thia. 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 offload protocol processing for normal data transfers from a host processor to a separate hardware and implement direct memory access by storing data (stripped of the TCP headers) to eliminate interlayer copying and improve performance. After the transport layer processing, the TCP header would be stripped off and the original byte stream would be transferred to host memory for use by the application.

B. Thia in Combination with Tanenbaum96 and further in Combination with Stevens2

133. A POSA would have been motivated to combine Stevens2 with Thia, in view of Tanenbaum96. Both Tanenbaum96 and Stevens2 were widely-referenced books describing different aspects of protocols such as TCP/IP. Tanenbaum96 references Stevens2 as providing a comprehensive treatment of TCP, IP and related protocols. Ex.1006, Tanenbaum96 at .790. Similarly,

Stevens2 expressly references the earlier 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 that was in a freely available and widely used BSD TCP/IP protocol stack implementation. Ex.1013, Stevens2 at .960-.963.

134. Stevens2 discloses that Header Prediction improves 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, Stevens2 at .962. In other words, calling for a reduced amount of protocol processing for the easiest and most common situation in the data transfer state. Ex.1006, Tanenbaum96 at .583 (“The key to fast TPDU processing is to separate out the normal case (one-way data transfer) and handle it specially.”). Header Prediction determines whether the next expected segment has been received by checking whether it is the same connection; the connection is ESTABLISHED; the SYN, FIN, RST, and URG flags are not set, and the segment is not out of order, consistent with the description of fast path processing in Stevens2. Ex.1013, Stevens2 at .962-.963. This is also similar to the description of the receive bypass test in Thia used to determine whether the received packet matches a template and is one in a normal data transfer phase. Ex.1015, Thia at

.003. This is not surprising because all three references are derived from or describing Jacobson’s header prediction algorithm.

135. 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 and free up the host processor to allow it to perform other necessary functions. This includes the interlayer copying that Thia seeks to avoid by performing this reduced operation protocol on its ROPE chip on the network interface adapter. Thus, a POSA would be motivated to use Thia’s separate hardware to offload processing from a host processor to efficiently process a continuous sequence of normal/expected packets that have a pre-established connection.

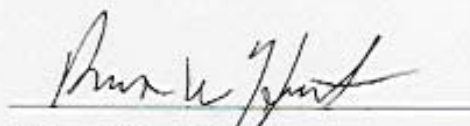
XI. GROUNDS OF INVALIDITY

136. I detail how the prior art invalidates the claims at issue in the Appendix A claim chart. In summary, my opinion is that claims 1, 3, 6-9, 11, 14-17, 19, and 21-22 of the 948 Patent are invalid over Thia in view of Tanenbaum96 and in further view of Stevens2.

Declaration

137. I declare that all statements made herein on my own knowledge are true and that all statements made on information and belief are believed to be true, and further, that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code.

Respectfully submitted,

A handwritten signature in black ink, appearing to read "Robert Horst", is written over a horizontal line.

Robert Horst, Ph.D

Date: 6/30/17

APPENDIX A
to Petition for IPR of U.S. 8,805,948

TABLE OF CONTENTS

	Page
[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	1
[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;	4
[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;	9
[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;	12
[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;	14
[1.5] if the first packet has any of the exception conditions, then protocol processing the first TCP header by the protocol processing stack;	16
[1.6] if the second packet has any of the exception conditions, then protocol processing the second TCP header by the protocol processing stack;	17
[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.	18

APPENDIX A
to Petition for IPR of U.S. 8,805,948

[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.....	24
[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.....	25
[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.	27
[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.	28
[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:	29
[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;	30
[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;	32
[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;	35
[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;	36

APPENDIX A
to Petition for IPR of U.S. 8,805,948

[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;	37
[9.6] if the second packet has any of the exception conditions, then protocol processing the second packet by the host computer;	38
[9.7] if the third packet has any of the exception conditions, then protocol processing the third packet by the host computer;	39
[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.....	40
[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.....	41
[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.	42
[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.....	43
[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.....	44
[17.P] An apparatus for network communication, the apparatus comprising:	45

APPENDIX A
to Petition for IPR of U.S. 8,805,948

[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;	47
[17.2.1] a network interface that is connected to the host computer by an input/output bus,.....	50
[17.2.2] the network interface adapted to parse the headers of received packets	51
[17.2.3] to determine whether the headers have the IP addresses and TCP ports that define the TCP connection and	52
[17.2.4] 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,	53
[17.2.5] the network interface having logic that directs any of the received packets that have the exception conditions to the protocol stack for processing, and.....	54
[17.2.6] [the network interface having logic that] 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.....	55
[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.	58
[21] The apparatus of claim 17, wherein the exception conditions include having a RST flag set.	60
[22] The apparatus of claim 17, wherein the exception conditions include having a SYN flag set.	61

Y.H. Thia & C.M. Woodside, A Reduced Operation Protocol Engine (ROPE) for a Multiple-layer Bypass Architecture (1995) (“Thia”) in view of Andrew S. Tanenbaum, Computer Networks (3rd ed. 1996) (“Tanenbaum96”) in further view of 2 Gary R. Wright & W. Richard Stevens, TCP/IP Illustrated: The Implementation (1995) (“Stevens2”)

[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

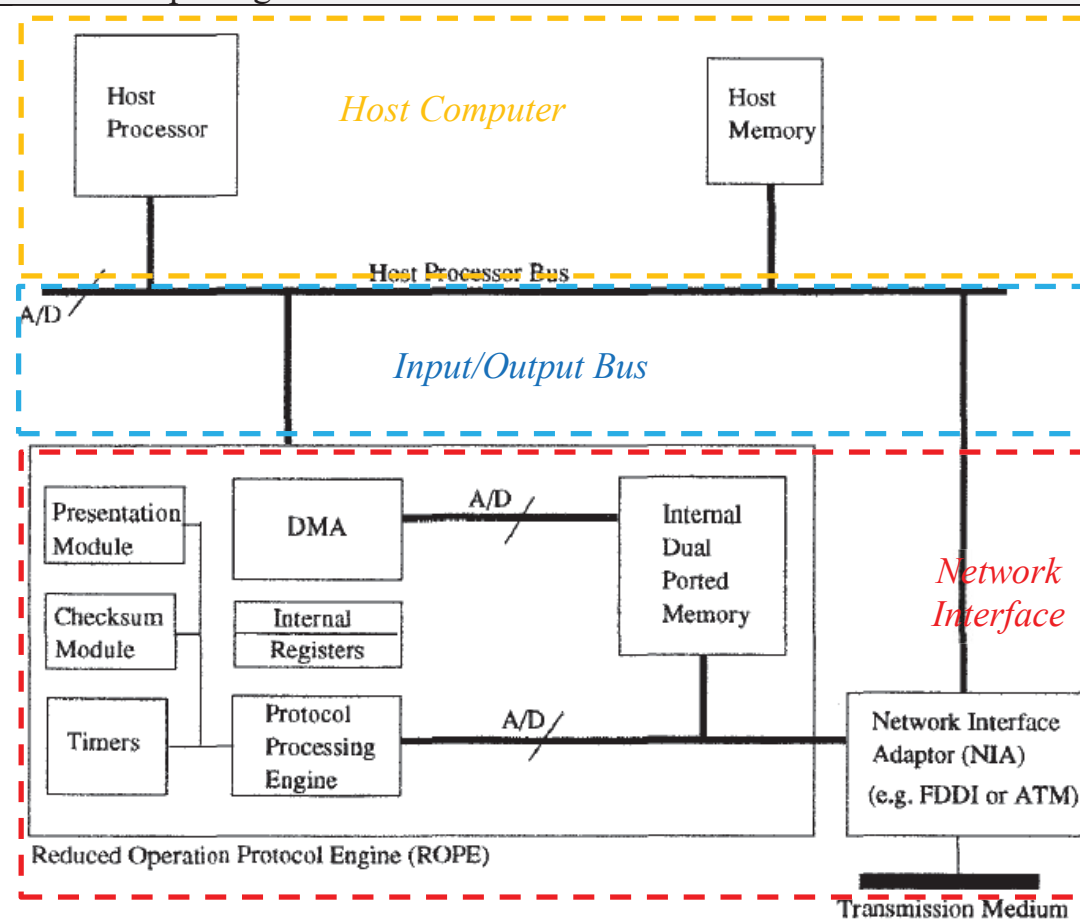
To the extent that the preamble is limiting, Thia in view of Tanenbaum96 in further view of Stevens2 discloses *a method for network communication by a host computer having a network interface that is connected to the host by an input/output bus.*

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*”). I annotate these components in Figure 2 of Thia below:

APPENDIX A
to Petition for IPR of U.S. 8,805,948

Y.H. Thia & C.M. Woodside, A Reduced Operation Protocol Engine (ROPE) for a Multiple-layer Bypass Architecture (1995) (“Thia”) in view of Andrew S. Tanenbaum, Computer Networks (3rd ed. 1996) (“Tanenbaum96”) in further view of 2 Gary R. Wright & W. Richard Stevens, TCP/IP Illustrated: The Implementation (1995) (“Stevens2”)

[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



Ex.1015, Thia at .007, Fig. 2.¹ A POSA would understand 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.

¹ Emphasis added unless otherwise noted.

APPENDIX A
to Petition for IPR of U.S. 8,805,948

Y.H. Thia & C.M. Woodside, A Reduced Operation Protocol Engine (ROPE) for a Multiple-layer Bypass Architecture (1995) (“Thia”) in view of Andrew S. Tanenbaum, Computer Networks (3rd ed. 1996) (“Tanenbaum96”) in further view of 2 Gary R. Wright & W. Richard Stevens, TCP/IP Illustrated: The Implementation (1995) (“Stevens2”)

[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

Accordingly, Thia in view of Tanenbaum96 in further view of Stevens2 discloses a method for network communication by *a* host processor and host memory (*host computer*) having a ROPE chip with the NIA (*network interface*) that is connected to the host processor by a Host Processor Bus (*input/output bus*).

Thia in view of Tanenbaum96 in further view of Stevens2

[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 view of Tanenbaum96 in further view of Stevens2 discloses *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.*

As explained in Section IX.A, 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, Thia at .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. I illustrated these components below:

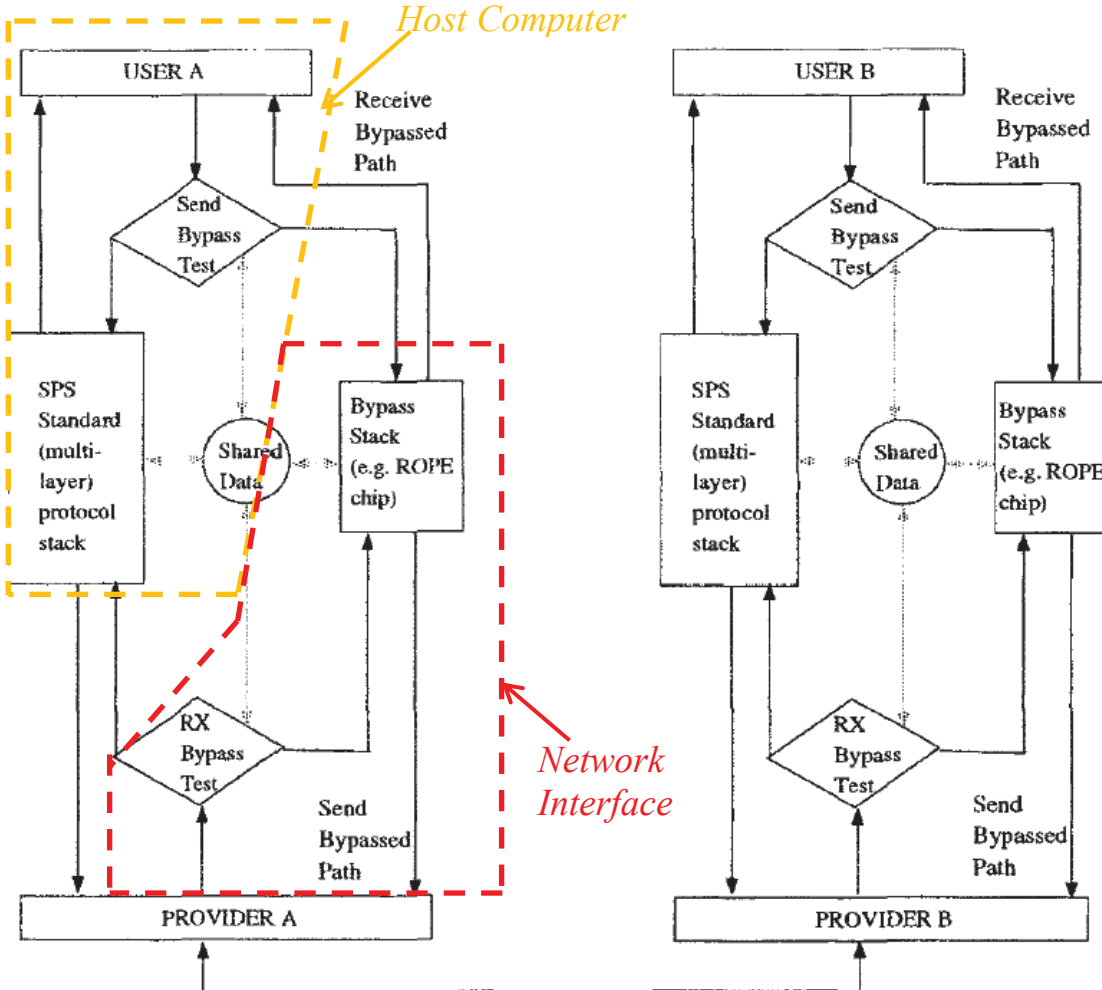


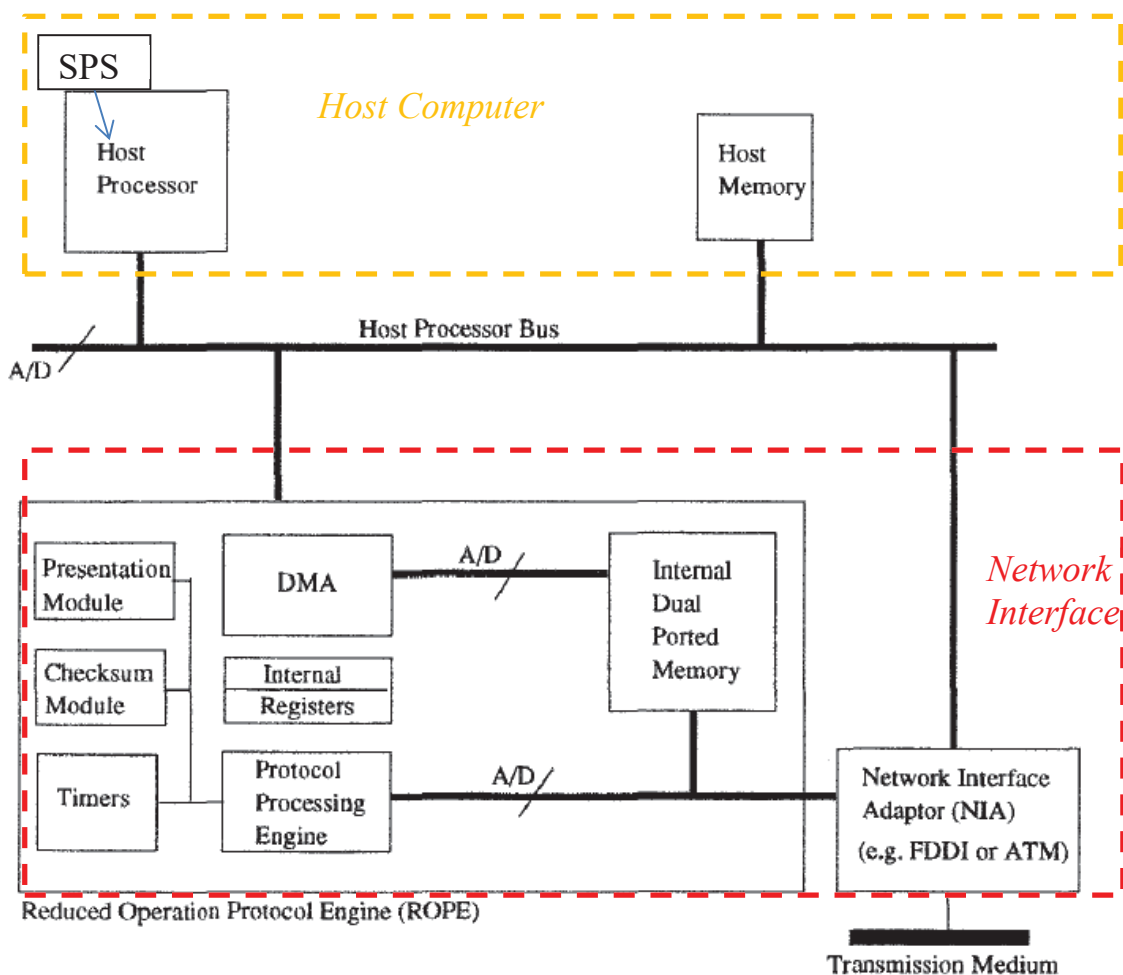
Figure 1 Bypass Architecture

APPENDIX A
to Petition for IPR of U.S. 8,805,948

Thia in view of Tanenbaum96 in further view of Stevens2

[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;

Ex.1015, Thia at .003, Fig.1.

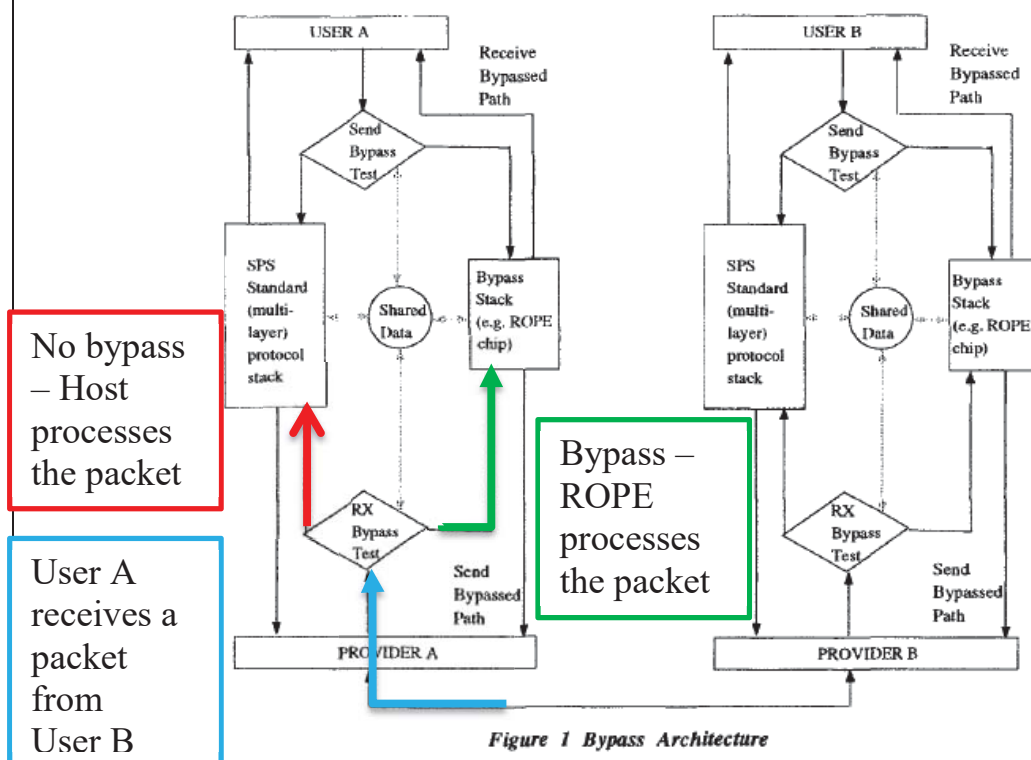


Ex.1015, Thia at .007, Fig.2.

APPENDIX A
to Petition for IPR of U.S. 8,805,948

Thia in view of Tanenbaum96 in further view of Stevens2

[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;



Ex.1015, Thia at .003, Fig.1; *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. . .”), .002 (“The contribution of this paper is to define the host/chip interface and the chip operation.”).

Thia discloses that the Standard Protocol Stack (SPS) protocol processes packets that are not bypassed. Ex.1015, Thia at .003 (“The standard protocol stack (SPS) is the processing path taken by all PDU’s during a connection without the bypass.”).

Thia in view of Tanenbaum96 discloses an *Internet Protocol (IP) layer and a Transmission Control Protocol (TCP) layer*. Thia discloses a hardware offload for the critical functions of a multiple-layer protocol stack (the SPS), and specifically discloses the design of a ROPE chip for the OSI Session and

APPENDIX A
to Petition for IPR of U.S. 8,805,948

Thia in view of Tanenbaum96 in further view of Stevens2

[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;

Transport layer protocols. Ex.1015, Thia at .001.

As explained in Sections X, 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, Tanenbaum96 at .054. Tanenbaum96 also discloses that TCP/IP became the dominant protocol suite as the OSI model vanished. *Id.* at .016. 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. Thus, the SPS would include an IP layer in place of the OSI Network layer and a TCP layer in place of the OSI Transport layer and the Session layer and presentation layer could be incorporated into the Application layer. *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.”). Thia’s bypass test is based on Jacobson’s prediction header for TCP/IP.

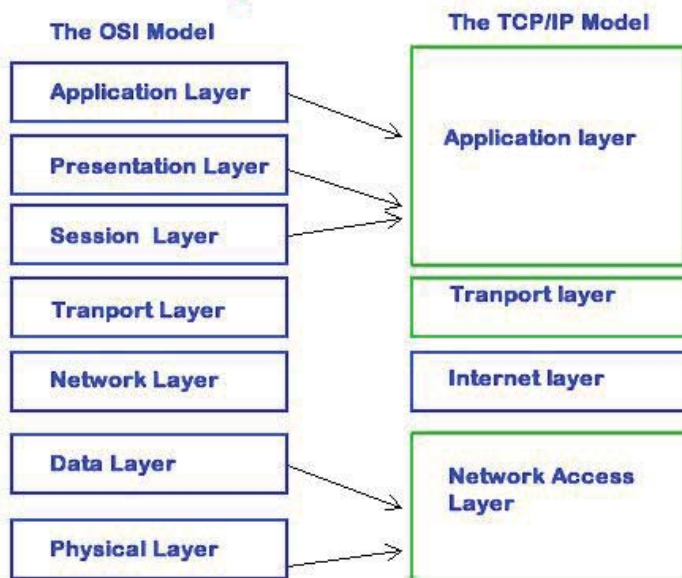
As shown in the figure below from Section V.A.2 above, the OSI and TCP/IP models have application layers above the transport layer.

APPENDIX A
to Petition for IPR of U.S. 8,805,948

Thia in view of Tanenbaum96 in further view of Stevens2

[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;

Contrasting the OSI and the TCP/IP Models



Accordingly, Thia in view of Tanenbaum96 in further view of Stevens2 discloses a host processor and host memory (together *a host computer*) with a Standard Protocol Stack (SPS) (*running 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 view of Tanenbaum96 in further view of Stevens2

[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;

Thia in view of Tanenbaum96 in further view of Stephens2 discloses *initializing, by the host computer, a TCP connection that is defined by source and destination IP addresses and source and destination TCP ports.*

Thia discloses initializing connections by a host. Ex.1015, Thia at .004. As I discussed above with respect to limitation [1.1], a POSA would have been motivated to implement Thia's Standard Protocol Stack (SPS) using the TCP/IP protocol suite.

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

APPENDIX A
to Petition for IPR of U.S. 8,805,948

This in view of Tanenbaum96 in further view of Stevens2

[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;

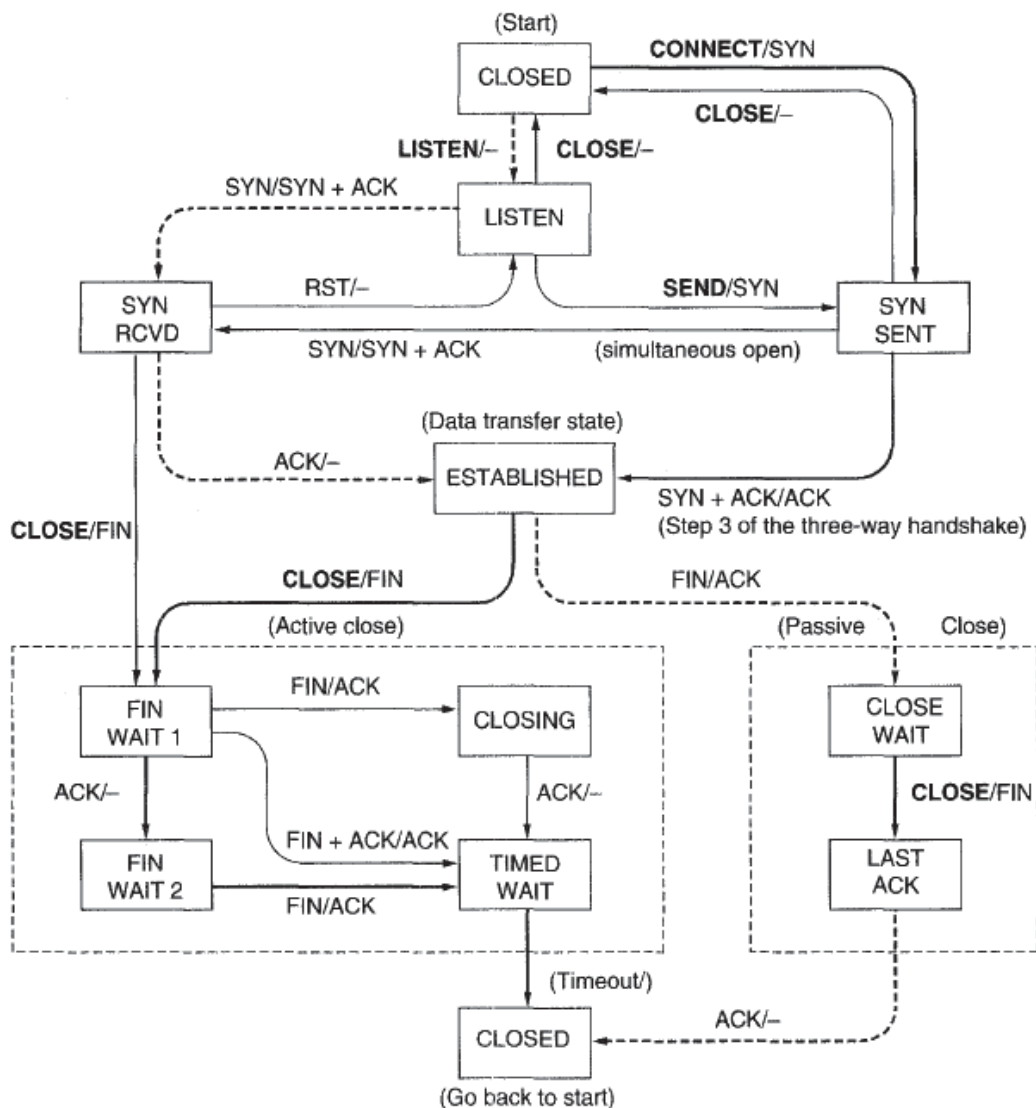


Fig. 6-28. TCP connection management finite state machine. The heavy solid line is the normal path for a client. The heavy dashed line is the normal path for a server. The light lines are unusual events.

Ex.1006, Tanenbaum96 at .550. The SYN flag is used to establish 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

APPENDIX A
to Petition for IPR of U.S. 8,805,948

Thia in view of Tanenbaum96 in further view of Stevens2

[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; initializing the TCP connection.

As explained in Sections V.B.4 and IX.B, a TCP/IP connection is defined by the IP address and TCP port for both sides of the connection. *See also* Ex.1006, Tanenbaum96 at .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 view of Tanenbaum96 in further view of Stevens2 discloses initializing, by the SPS on the host processor (*host computer*), a TCP connection that is defined by source and destination IP addresses and source and destination TCP ports.

Thia in view of Tanenbaum96 in further view of Stevens2

[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;

Thia in view of Tanenbaum96 in further view of Stevens2 discloses *a communication processing mechanism connected to the first processor.*

As I explained for limitation [1.P], Thia's Network Interface Adapter (NIA), together with the ROPE chip, is the *network interface*. As explained in Section IX.A, Thia's NIA receives multiple packets, including consecutive packets:

2.1 Bypass Architecture

Figure 1 illustrates the architecture of a bypass implementation for any standard protocol. The standard protocol stack (SPS) is the processing path taken by all PDUs [Protocol Data Units i.e. packets] during a connection without the bypass.... 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. The shared data are used to maintain state consistency between the SPS and the bypass stack, including window flow control parameters and connection identifiers.

Ex.1015, Thia at .003.

As disclosed in Tanenbaum96, a network interface connected to a TCP/IP network (as explained above for limitation [1.1]) obviously receives TCP/IP packets (segments). TCP/IP packets contain TCP headers and optionally payload data. The well-known structure of TCP/IP packets with TCP Headers and payload data is documented in Tanenbaum96. *See e.g.*, Ex.1006, Tanenbaum96 at .544-.547. The TCP header is illustrated below:

APPENDIX A
to Petition for IPR of U.S. 8,805,948

Thia in view of Tanenbaum96 in further view of Stevens2

[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;

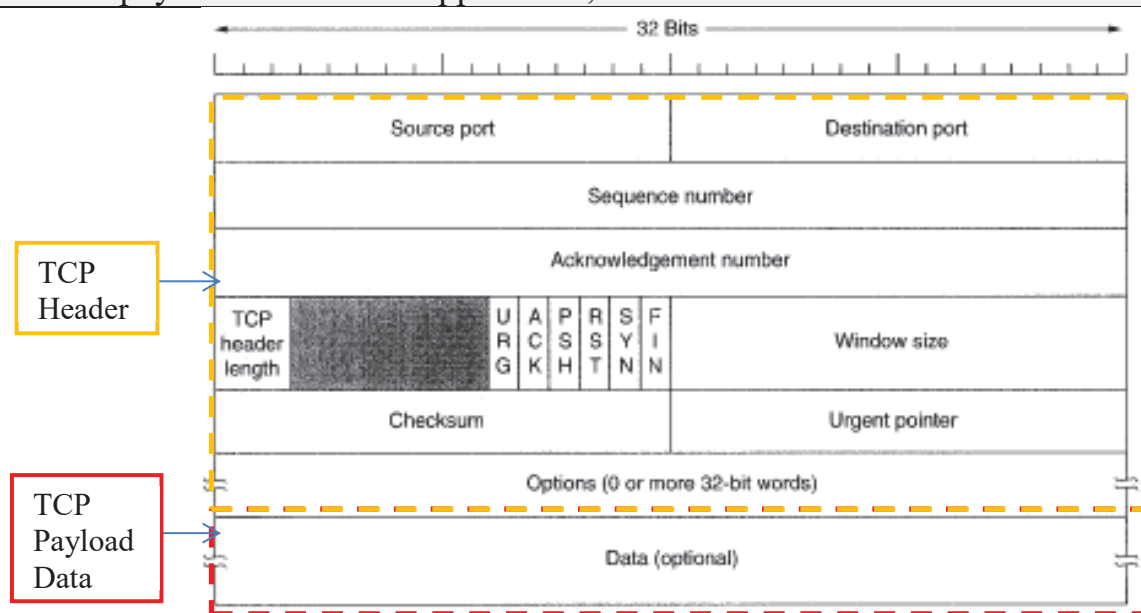


Fig. 6-24. The TCP header.

Id. at .544. (Annotated)

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

As explained above for claim limitation [1.1], the application layer is above the Transport layer in TCP/IP and OSI model implementations. Ex.1006, Tanenbaum96 at .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*).

Thus, Thia in view of Tanenbaum96 in further view of 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*).

APPENDIX A
to Petition for IPR of U.S. 8,805,948

Thia in view of Tanenbaum96 in further view of Stevens2

[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;

Thia in view of Tanenbaum96 in further view of Stevens2 discloses *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 explained in Section IX.A, Thia's NIA (part of the *network interface*) performs the receive bypass test (*checking*).

As explained in Sections X.A and X.B, 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 well-known "Header Prediction algorithm" for TCP/IP, which is also described in the Tanenbaum96 and Stevens2 references discussed below. Ex.1015, Thia at .002. As explained in Tanenbaum96 and Stevens2, 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*). See Ex.1006, Tanenbaum96 at .565, .566-.567 (many TCP implementations use it); .585 (disclosing the test above), .545 (description of flags), .542 (urgent flag); *see also* Ex.1013, Stevens2 at .962-.963 (walkthrough of the BSD code for the test above, which tests whether control flags SYN, FIN, RST, or URG are set). Thia discloses that bypass processing is for bulk data transfer and excludes reassembly of fragmented packets, which should be restricted to the lower layers. Ex.1015, Thia at .002, .014.

Thus, Thia in view of Tanenbaum96 in further view of Stevens2 discloses the NIA (part of the *network interface*) performing a receive bypass test (*checking . . . whether the packets have certain exception conditions*)

APPENDIX A
to Petition for IPR of U.S. 8,805,948

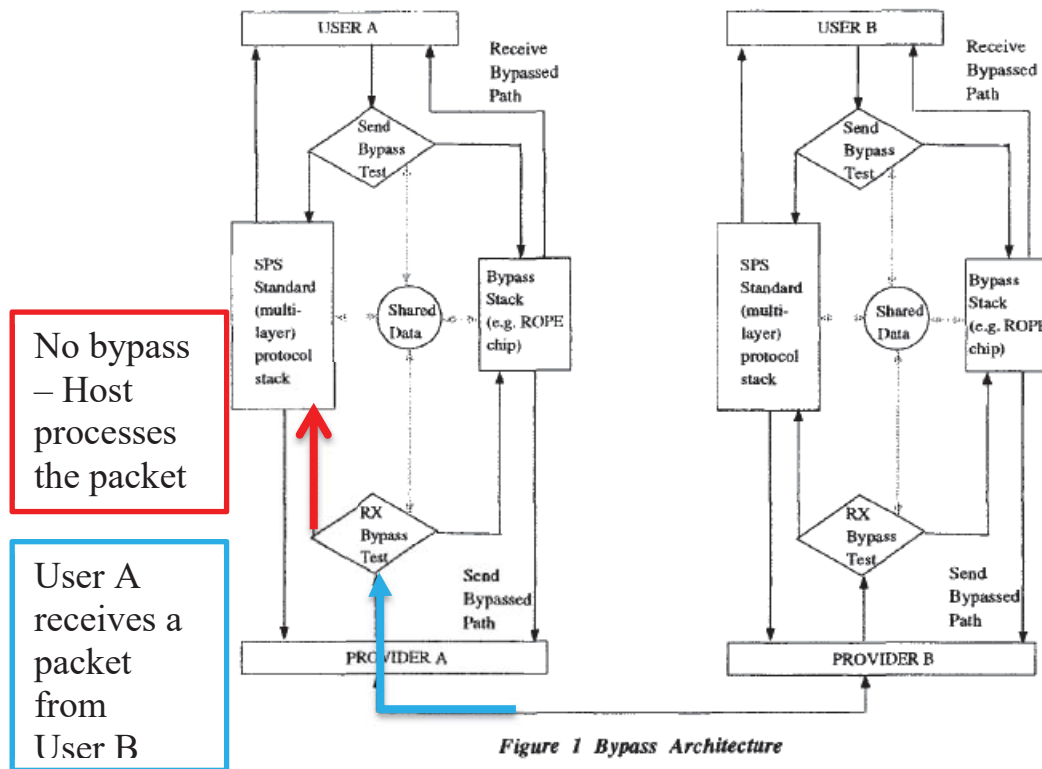
Thia in view of Tanenbaum96 in further view of Stevens2
[1.4] checking, by the network interface, whether the packets have certain exception conditions, including checking whether the packets are IP fragmented, checking whether the packets have a FIN flag set, and checking whether the packets are out of order;
<i>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.</i>

Thia in view of Tanenbaum96 in further view of Stevens2

[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 view of Tanenbaum96 in further view of 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 IX.A, Thia teaches that packets that fail the receive bypass test are processed by the Standard Protocol Stack (SPS) (*the protocol processing stack*).



Ex.1015, Thia at .003 (Bypass Architecture), Fig.1. Thus, packets failing the conditions of Header Prediction would fail the receive bypass test and be processed by the SPS.

Thus, Thia in view of Tanenbaum96 in further view of Stevens2 discloses that *if the first packet has any of the exception conditions* (failing the conditions of Header Prediction), *then protocol processing the first TCP header by the protocol processing stack* (Standard Protocol Stack).

Thia in view of Tanenbaum96 in further view of Stevens2

[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 view of Tanenbaum96 in further view of Stevens2 discloses if the second packet has any of the exception conditions, then protocol processing the second TCP header by the protocol processing stack.

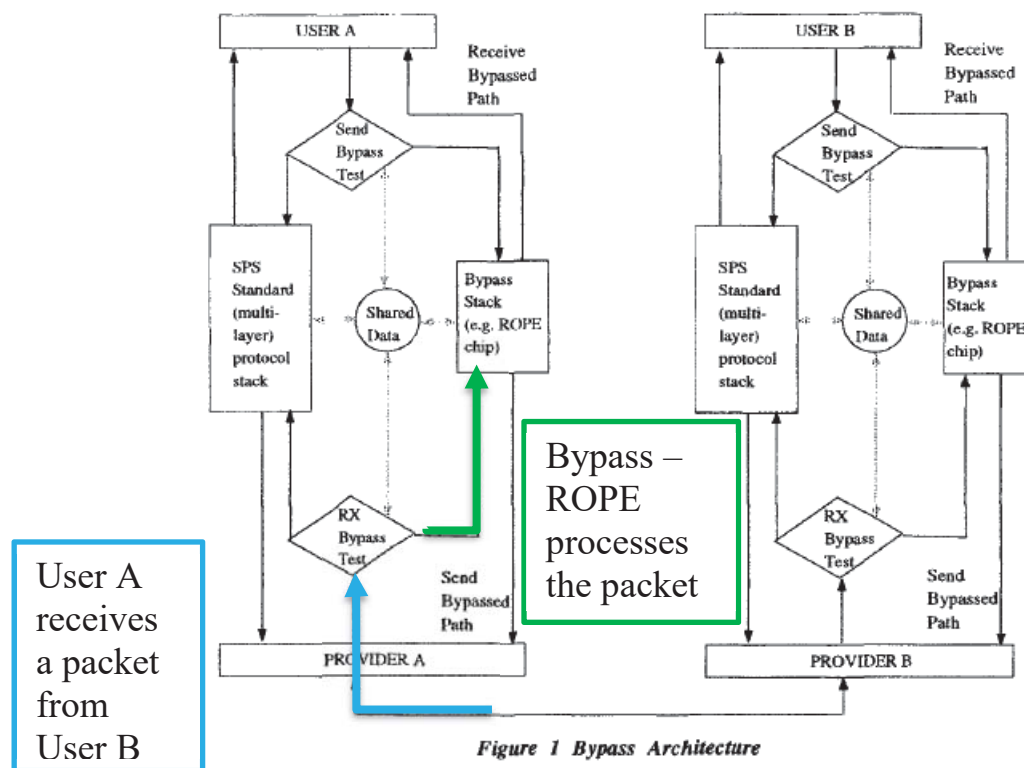
The same receive bypass test applies to all packets. Ex.1015, Thia at .003 (“The receive bypass test matches the incoming PDU headers with a template that identifies the predicted bypassable headers.”). Thus, *see* claim limitation [1.5] above.

Accordingly, Thia in view of Tanenbaum96 in further view of Stevens2 discloses that *if the second packet has any of the exception conditions* (failing the conditions of Header Prediction), *then protocol processing the first TCP header by the protocol processing stack* (Standard Protocol Stack).

[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.

Thia in view of Tanenbaum96 in further view of Stevens2 discloses 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 explained in Section IX.A, 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. I have illustrated this in Figure 1 of Thia below:



Ex.1015, Thia at .003, Fig.1 (annotated).

APPENDIX A
to Petition for IPR of U.S. 8,805,948

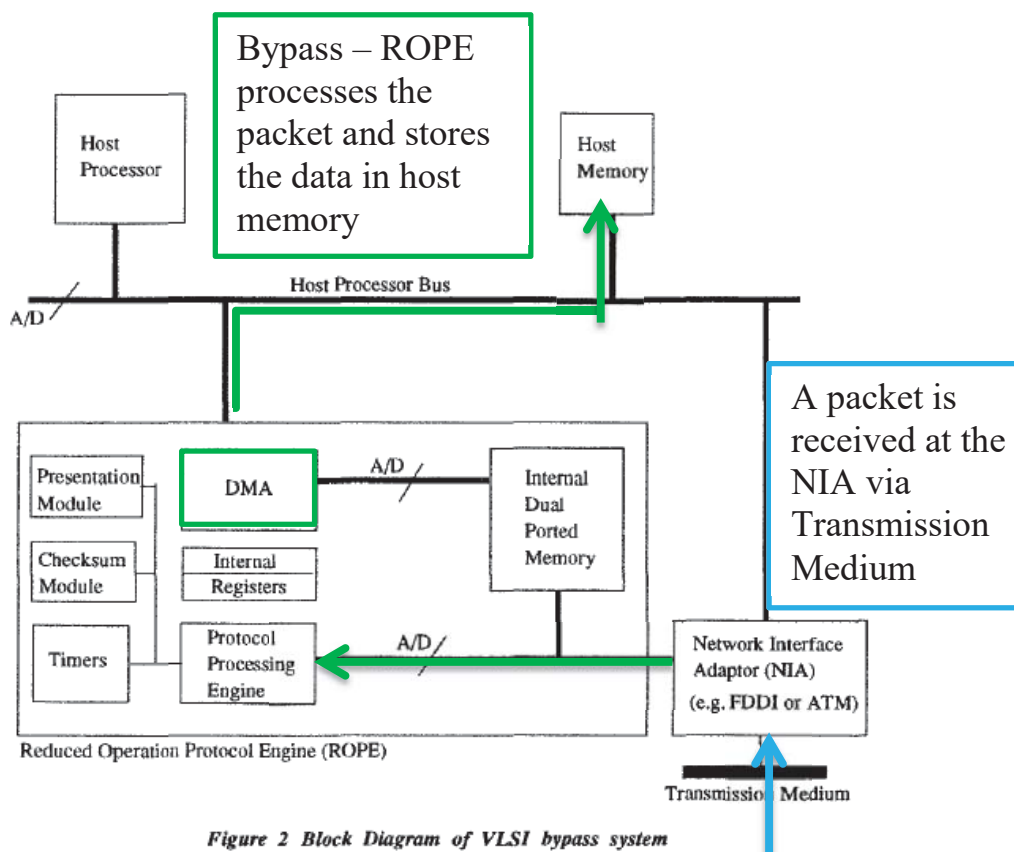
Thia in view of Tanenbaum96 in further view of Stevens2

[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.

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. I have illustrated this in Figure 2 of Thia below:



Ex.1015, Thia at .007. 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*).

APPENDIX A
to Petition for IPR of U.S. 8,805,948

Thia in view of Tanenbaum96 in further view of Stevens2

[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.

Thia in view of Tanenbaum96 in further view of 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. As I've highlighted in Table 1 of Thia, the Header Decode function occurs on the ROPE chip and data copying within layers is eliminated:

APPENDIX A
to Petition for IPR of U.S. 8,805,948

Thia in view of Tanenbaum96 in further view of Stevens2

[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.

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

Further, as taught by Tanenbaum, by the time the data reaches the application in host memory, it is stored as a byte stream. Tanenbaum96

APPENDIX A
to Petition for IPR of U.S. 8,805,948

Thia in view of Tanenbaum96 in further view of Stevens2

[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.

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, Tanenbaum96 at .498 ("The transport entity can be . . . on the network interface card"), at .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.

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. 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 Direct Memory Access (DMA) to transfer only the data blocks in order to a buffer in host memory (*storing . . . together in a buffer of the host computer*).

APPENDIX A
to Petition for IPR of U.S. 8,805,948

Thia in view of Tanenbaum96 in further view of Stevens2

[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.

Accordingly, Thia in view of Tanenbaum96 in further view of Stevens2 discloses that failing the conditions of Header Prediction (*if the packets have any of the exception conditions*), the Standard Protocol Stack processing of the headers on the host is bypassed (*bypassing host protocol processing of the TCP headers*) and the data is stored together in Host Memory after the header processing is done by the ROPE (*storing the first payload data and the second payload data together in a buffer in order and without any TCP header stored between the first payload data and the second payload data*).

APPENDIX A
to Petition for IPR of U.S. 8,805,948

Thia in view of Tanenbaum96 in further view of Stevens2

[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 view of Tanenbaum96 in further view of Stevens2 discloses 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.

As explained for claim limitation [1.8], 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, Thia at 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.1006, Tanenbaum96 at .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, Thia at Table1, .002, .006, .007.

Accordingly, Thia in view of Tanenbaum96 in further view of Stevens2 discloses storing consecutive packets (*wherein storing the first payload data and the second payload data*) in Host memory (*together in a buffer of the host computer*) is performed by the DMA (*is performed by a direct memory access (DMA) unit*) on the ROPE (*of the network interface*).

APPENDIX A
to Petition for IPR of U.S. 8,805,948

Thia in view of Tanenbaum96 in further view of Stevens2

[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 view of Tanenbaum96 in further view of Stevens2 discloses 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.

As explained in Section IX.A, 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, Thai at .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, Tanenbaum96 at .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* above discussion of claim element [1.1] regarding the TCP/IP connection being defined by source and destination IP addresses and source and destination TCP ports.

In combining Thia with Tanenbaum96 for TCP/IP offload, a POSA would have been motivated to use a template to do the comparison, such as the prototype header provided by Tanenbaum96, for the receive bypass test. As I've illustrated below, this template includes source and destination IP addresses and source and destination TCP Ports.

APPENDIX A
to Petition for IPR of U.S. 8,805,948

Thia in view of Tanenbaum96 in further view of Stevens2

[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.

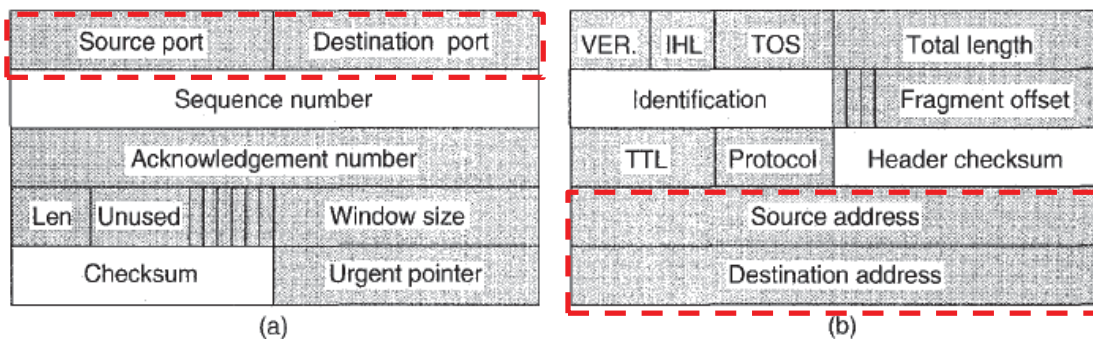


Fig. 6-50. (a) TCP header. (b) IP header. In both cases, the shaded fields are taken from the prototype without change.

Ex.1006, Tanenbaum96 at .584.

Accordingly, Thia in view of Tanenbaum96 in further view of Stevens2 discloses *comparing, by the networking 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* (prototype header template).

APPENDIX A
to Petition for IPR of U.S. 8,805,948

Thia in view of Tanenbaum96 in further view of Stevens2

[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 view of Tanenbaum96 in further view of Stevens2 discloses the method of claim 1, wherein checking whether the packets have certain exception conditions includes checking whether the packets have a RST flag set.

As explained for claim limitation [1.4], Thia's NIA (part of the *network interface*) checks the packets for certain exception conditions as part of its bypass test. As explained in Sections X.A and X.B, 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 that is based on Jacobson's disclosure. As explained in Section IX.C, Jacobson's Header Prediction test determines whether the RST flag is set. Ex.1013, Stevens2 at .962 ("The following four control flags must not be on: SYN, FIN, RST, or URG.")

Accordingly, Thia in view of Tanenbaum96 in further view of Stevens2 discloses the use of Header Prediction that checks whether certain flags are set (*wherein checking whether the packets have certain exception conditions*) including whether the RST flag is set (*includes checking whether the packets have a RST flag set*).

APPENDIX A
to Petition for IPR of U.S. 8,805,948

Thia in view of Tanenbaum96 in further view of Stevens2

[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 view of Tanenbaum96 in further view of Stevens2 discloses the method of claim 1, wherein checking whether the packets have certain exception conditions includes checking whether the packets have a SYN flag set.

As explained for claim limitation [1.4], Thia's NIA (part of the *network interface*) checks the packets for certain exception conditions as part of its bypass test. As explained in Sections X.A and X.B, a POSA would have been motivated to implement the Jacobson-based Header Prediction disclosed in Tanenbaum and Stevens2 as a part of Thia's receive bypass test that is based on Jacobson's disclosure. As explained in Section IX.C, Jacobson's Header Prediction test determines whether the SYN flag is set. Ex.1013, Stevens2 at .962("The following four control flags must not be on: WYN, FIN, RST, or URG.")

Accordingly, Thia in view of Tanenbaum96 in further view of Stevens2 discloses the use of Header Prediction that checks whether certain flags are set (*wherein checking whether the packets have certain exception conditions*) including whether the RST flag is set (*includes checking whether the packets have a RST flag set*).

APPENDIX A
to Petition for IPR of U.S. 8,805,948

Thia in view of Tanenbaum96 in further view of Stevens2
<p>[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:</p>
<p>As explained with respect to claim limitation [1.P] for the identical preamble of Claim 1, to the extent that the preamble is limiting, Thia in view of Tanenbaum96 in further view of 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.</p>

APPENDIX A
to Petition for IPR of U.S. 8,805,948

Thia in view of Tanenbaum96 in further view of Stevens2

[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 view of Tanenbaum96 in further view of Stevens2 discloses 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.

As explained above for claim limitation [1.P], Thia's NIA, together with the ROPE chip, is the *network interface*. As explained in Section IX.A, the NIA receives consecutive packets. A network interface connected to a TCP/IP network (as explained above for claim limitation [1.2]) receives TCP/IP packets.

As explained above for claim limitation [1.3], Tanenbaum teaches that TCP/IP packets have TCP headers. Therefore, the first packet has a TCP header.

As explained above for claim limitation [6], a TCP/IP packet header contains Source and Destination IP addresses and Source and Destination Ports.

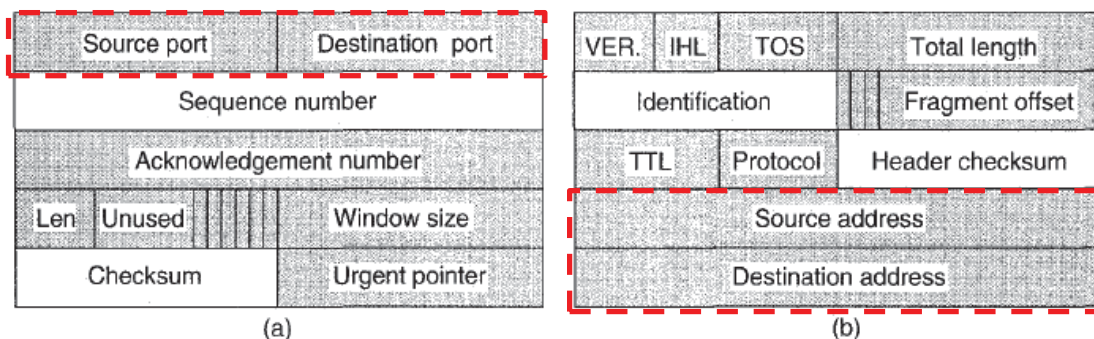


Fig. 6-50. (a) TCP header. (b) IP header. In both cases, the shaded fields are taken from the prototype without change.

Ex.1006, Tanenbaum96 at .584.

Accordingly, Thia in view of Tanenbaum96 in further view of Stevens2 discloses that the NIA and ROPE receive multiple packets including a first packet (*receiving, by the network interface, a first packet*) and when

APPENDIX A
to Petition for IPR of U.S. 8,805,948

Thia in view of Tanenbaum96 in further view of Stevens2
[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;
connected to a TCP/IP network, the packets include a header with source and destination Internet Protocol (IP) addresses and source and destination Transmission Control Protocol (TCP) ports.

APPENDIX A
to Petition for IPR of U.S. 8,805,948

Thia in view of Tanenbaum96 in further view of Stevens2

[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 view of Tanenbaum96 in further view of Stevens2 discloses 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.

As explained above for claim limitation [1.1], Thia in view of Tanenbaum96 in further view of 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 for claim limitation [1.2], Thia in view of Tanenbaum96 in further view of 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*).

APPENDIX A
to Petition for IPR of U.S. 8,805,948

This in view of Tanenbaum96 in further view of Stevens2

[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;

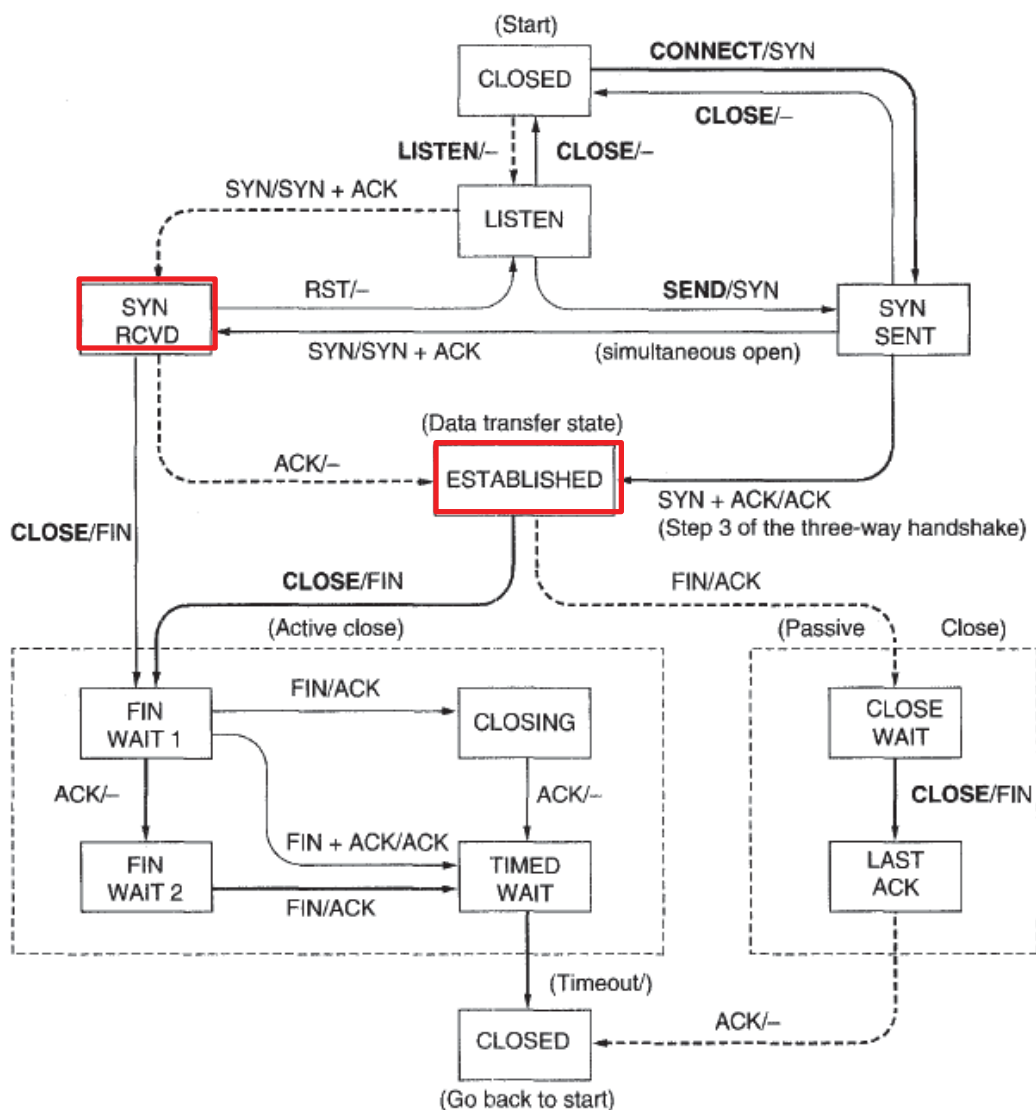


Fig. 6-28. TCP connection management finite state machine. The heavy solid line is the normal path for a client. The heavy dashed line is the normal path for a server. The light lines are unusual events.

Ex.1006, Tanenbaum96 at .550.

As explained above for claim limitation [1.2], a TCP/IP connection is defined by the IP address and TCP port for both sides of the connection.

APPENDIX A
to Petition for IPR of U.S. 8,805,948

Thia in view of Tanenbaum96 in further view of Stevens2

[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;

Accordingly, Thia in view of Tanenbaum96 in further view of Stevens2 discloses that the SPS on the host (*protocol processing, by the host computer, the first packet*) processes a packet intended to establish a connection, such as a SYN packet to initialize a TCP/IP connection, by initializing a connection (*thereby initializing a TCP connection*). When the connection is a TCP/IP connection, that connection is defined by the source and destination IP addresses and source and destination TCP ports.

APPENDIX A
to Petition for IPR of U.S. 8,805,948

Thia in view of Tanenbaum96 in further view of Stevens2

[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 view of Tanenbaum96 in further view of Stevens2 discloses 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.

The first packet in claim limitation [1.3] discussed above corresponds to the second packet in [9.3] because Claim 9 introduces an earlier packet as part of the initialization. As explained above for claim limitation [1.3], Thia's NIA receives packets and is part of *the network interface*. As also explained above for claim limitation [1.3], modifying Thia to use the TCP/IP packets, the received TCP/IP packets contain headers and optionally contain payload data.

Claim 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 Sections V.B, V.D.4 and IX.B, when a connection is initialized, a connection record is recorded containing the source and destination IP addresses and source and destination TCP ports. As explained above for claim limitation [6], Thia's modified bypass test searches for the 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.

Accordingly, Thia in view of Tanenbaum96 in further view of Stevens2 discloses receiving a second packet after initialize the connection with the first packet (*receiving, by the network interface, a second packet*) and when connected to a TCP network, that second packet has a second header and payload data. If the second packet is for the same TCP connection as the first packet, the second header will have IP addresses and TCP ports that match IP addresses and TCP ports of the TCP connection.

APPENDIX A
to Petition for IPR of U.S. 8,805,948

Thia in view of Tanenbaum96 in further view of Stevens2
<p>[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;</p>
<p>Thia in view of Tanenbaum96 in further view of Stevens2 discloses 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.</p>
<p>The limitations of [9.4] are identical to [9.3], but for them being in the context of a third packet. As I discussed above in Sections X.A and X.B, TCP network interface generally receives multiple TCP packets for the same TCP connection (e.g. data transfer phase in Thia and Tanenbaum). Thus, the limitations explained above in Section 0 also disclose this limitation.</p>

APPENDIX A
to Petition for IPR of U.S. 8,805,948

Thia in view of Tanenbaum96 in further view of Stevens2

[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;

Thia in view of Tanenbaum96 in further view of Stevens2 discloses 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 of [9.5] are substantially identical to [1.4] as both relate to checking multiple packets; [1.4] checks “the packets” whereas [9.5] checks “the second and third packets.” As explained above for claim limitation [1.4], 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.

APPENDIX A
to Petition for IPR of U.S. 8,805,948

Thia in view of Tanenbaum96 in further view of Stevens2
[9.6] if the second packet has any of the exception conditions, then protocol processing the second packet by the host computer;
Thia in view of Tanenbaum96 in further view of 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 for claim limitation [9.3], the first packet in claim 1 corresponds to the second packet in claim 9 because claim 9 introduces an earlier packet as part of the initialization. As explained for claim limitation [1.5], packets with any of the exception conditions fail Thia's receive bypass test and are protocol processed by the SPS.

APPENDIX A
to Petition for IPR of U.S. 8,805,948

Thia in view of Tanenbaum96 in further view of Stevens2
[9.7] if the third packet has any of the exception conditions, then protocol processing the third packet by the host computer;
Thia in view of Tanenbaum96 in further view of Stevens2 discloses if the third packet has any of the exception conditions, then protocol processing the third packet by the host computer.
The same receive bypass test in Thia applies to all packets. Thus, <i>see</i> claim limitation [9.6] above.

APPENDIX A
to Petition for IPR of U.S. 8,805,948

Thia in view of Tanenbaum96 in further view of Stevens2
<p>[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.</p>
<p>Thia in view of Tanenbaum96 in further view of Stevens2 discloses this limitation.</p> <p>The claim limitation [1.7] includes the claim limitations of [9.8]. In both 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. Claim limitation [1.7] has an additional limitation of bypassing host protocol processing of the TCP headers, which is not found in claim limitation [9.8]. Regardless, the disclosure satisfying claim limitation [1.7] also satisfies claim limitation [9.8]. Thus, <i>see</i> Claim limitations [1.7].</p>

APPENDIX A
to Petition for IPR of U.S. 8,805,948

Thia in view of Tanenbaum96 in further view of Stevens2
<p>[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.</p>
<p>Thia in view of Tanenbaum96 in further view of Stevens2 discloses 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.</p> <p>This claim is substantially similar to claim 3. 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. Thus, <i>see</i> claim limitation [3].</p>

APPENDIX A
to Petition for IPR of U.S. 8,805,948

Thia in view of Tanenbaum96 in further view of Stevens2

[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 view of Tanenbaum96 in further view of Stevens2 discloses 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.

This claim is substantially similar to claim 6. Claim 6 requires “comparing . . . the IP addresses and TCP ports of the packets” wherein packets refers 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. Thus, *see* claim limitation [6].

APPENDIX A
to Petition for IPR of U.S. 8,805,948

Thia in view of Tanenbaum96 in further view of Stevens2

[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 view of Tanenbaum96 in further view of 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 RST flag set.

This claim is substantially similar to claim 7. 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. Thus, *see* claim limitation [7].

APPENDIX A
to Petition for IPR of U.S. 8,805,948

Thia in view of Tanenbaum96 in further view of Stevens2

[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 view of Tanenbaum96 in further view of 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. 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. Thus, *see* claim limitation [8].

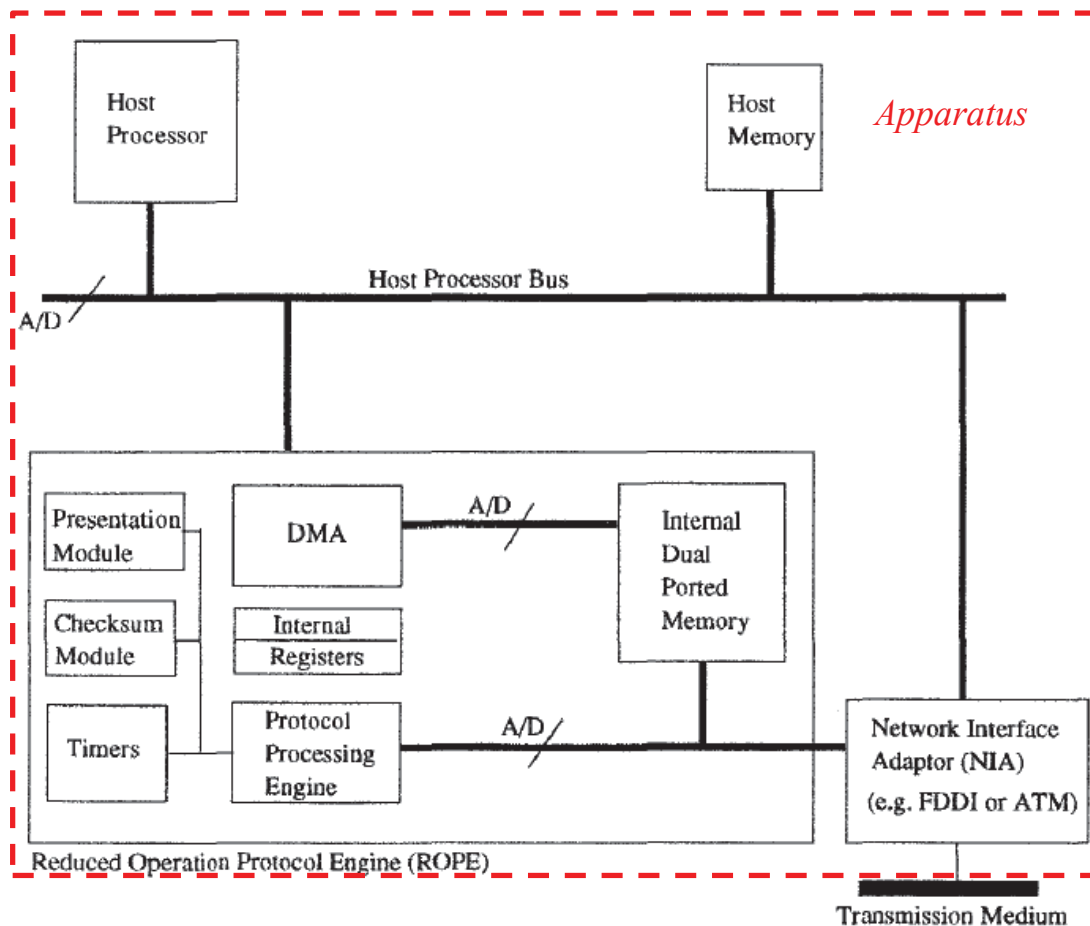
APPENDIX A
to Petition for IPR of U.S. 8,805,948

Thia in view of Tanenbaum96 in further view of Stevens2

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

To the extent that the preamble is limiting, Thia in view of Tanenbaum96 in further view of 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. I have illustrated these components in Figure 2 of Thia below:



Ex.1015, Thia at .007.

Thia discloses that the NIA (portion of the *apparatus*) is used for sending and receiving packets (*network communications*). Ex.1015, Thia at .008

APPENDIX A
to Petition for IPR of U.S. 8,805,948

Thia in view of Tanenbaum96 in further view of Stevens2
[17.P] An apparatus for network communication, the apparatus comprising:
(describing the NIA as a source/sink for data packets.)
Accordingly, Thia in view of Tanenbaum96 in further view of Stevens2 discloses an apparatus for network communications.

APPENDIX A
to Petition for IPR of U.S. 8,805,948

Thia in view of Tanenbaum96 in further view of Stevens2

[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;

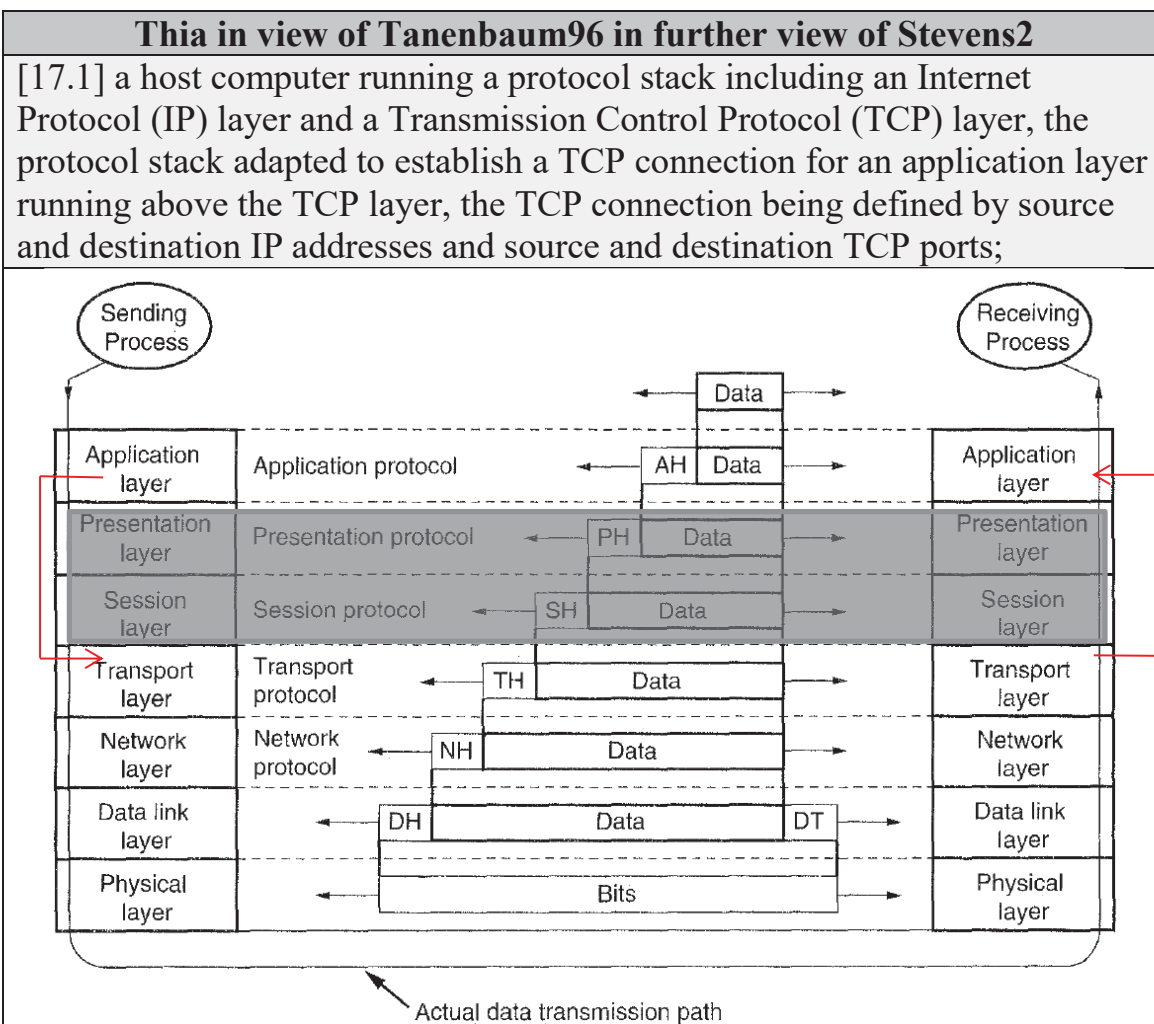
Thia in view of Tanenbaum96 in further view of Stevens2 discloses this limitation.

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.

As explained above for claim limitation [1.1], Thia in view of Tanenbaum96 in view of Stevens2 discloses an SPS running on the host (*a host computer running a protocol stack*). As also explained above for claim limitation [1.1], Thia in view of Tanenbaum96 in view of 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 for claim limitation [1.2], Thia in view of Tanenbaum96 in view of 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*).

APPENDIX A
to Petition for IPR of U.S. 8,805,948



Ex.1006, Tanenbaum96 at .052-.053, Fig.1-17.

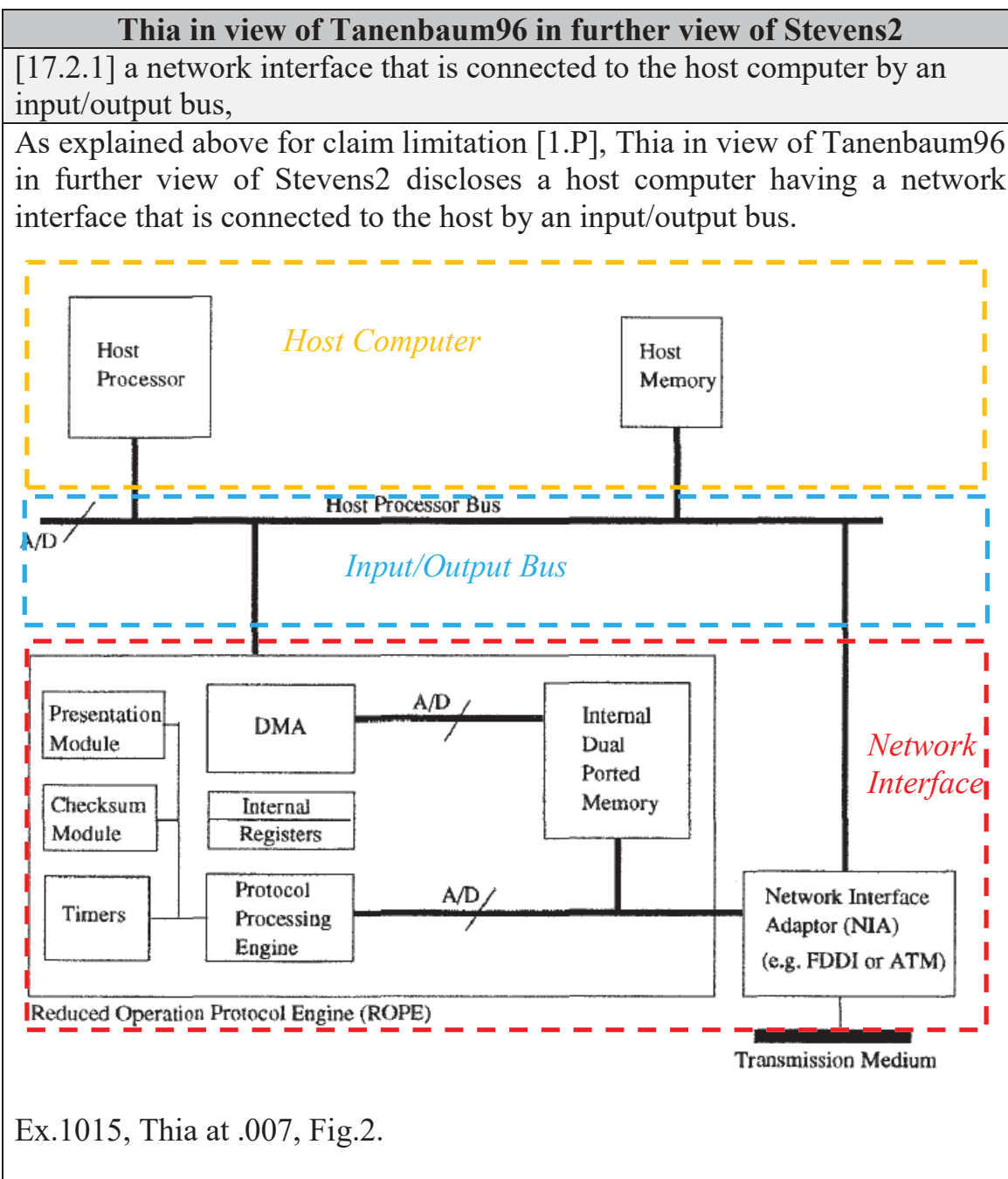
As explained above for claim limitation [1.3], Thia in view of Tanenbaum96 in view of Stevens2 discloses that the TCP connection is defined by the TCP ports and the IP addresses for both sides of the connection (*the TCP connection being defined by source and destination IP addresses and source and destination TCP ports*).

Accordingly, Thia in view of Tanenbaum96 in further view of Stevens2 discloses a host computer running a Standard Protocol Stack (*protocol stack*) and when connected to a TCP connection, the protocol stack would include an IP layer and a TCP layer and would establish a TCP connection for an application layer running above the TCP layer. The TCP connection would be defined by the source and destination IP addresses and source and

APPENDIX A
to Petition for IPR of U.S. 8,805,948

Thia in view of Tanenbaum96 in further view of Stevens2
[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;
destination TCP ports.

APPENDIX A
to Petition for IPR of U.S. 8,805,948



APPENDIX A
to Petition for IPR of U.S. 8,805,948

Thia in view of Tanenbaum96 in further view of Stevens2
[17.2.2] 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. Thia discloses a receive bypass test (performed by the NIA, part of <i>the network interface</i>) in which fields of the header of a received packet are compared to a template. Ex.1015, Thia at .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.

APPENDIX A
to Petition for IPR of U.S. 8,805,948

Thia in view of Tanenbaum96 in further view of Stevens2
[17.2.3] to determine whether the headers have the IP addresses and TCP ports that define the TCP connection and
As explained for claim limitation [6]. Thia in view of Tanenbaum96 in further view of Stevens2 discloses the NIA performing a modified receive bypass test that checks whether the packet matches a template (i.e., has source and destination IP addresses and source and destination TCP ports that match the connection record). In doing so, the NIA determines <i>whether the headers have the IP addresses and TCP ports that define the TCP connection</i> .

APPENDIX A
to Petition for IPR of U.S. 8,805,948

This in view of Tanenbaum96 in further view of Stevens2

[17.2.4] 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,

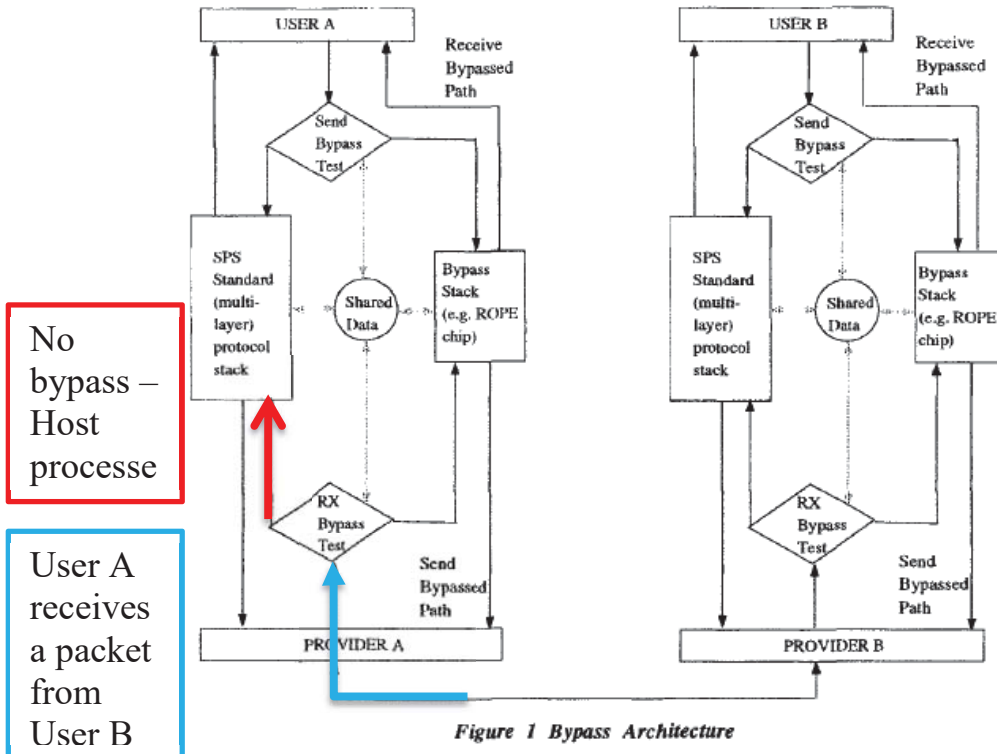
As explained for claim limitation [1.4] the NIA (part of *the network interface*) performs a modified receive bypass test as taught by Tanenbaum96 on received packets to check whether each TCP header has an established connection, neither side is trying to close the connection (i.e., *no FIN or RST flag*), 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, Tanenbaum96 at .585 (disclosing the test above); *see also* Ex.1013, Stevens2 at .962-.963 (providing a walkthrough of the BSD code for the test above).

APPENDIX A
to Petition for IPR of U.S. 8,805,948

Thia in view of Tanenbaum96 in further view of Stevens2

[17.2.5] the network interface having logic that directs any of the received packets that have the exception conditions to the protocol stack for processing, and

As explained above for claim limitation [1.5], Thia in view of Tanenbaum96 in further view of Stevens2 discloses that packets that fail the bypass test are processed by the SPS.



Ex.1015, Thia at .003, Fig.1.

to Petition for IPR of U.S. 8,805,948

Thia in view of Tanenbaum96 in further view of Stevens2

[17.2.6] [the network interface having logic that] 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 for claim limitation [1.7], Thia in view of Tanenbaum96 in further view of 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. I have illustrated this path in Figure 1 of Thia below:

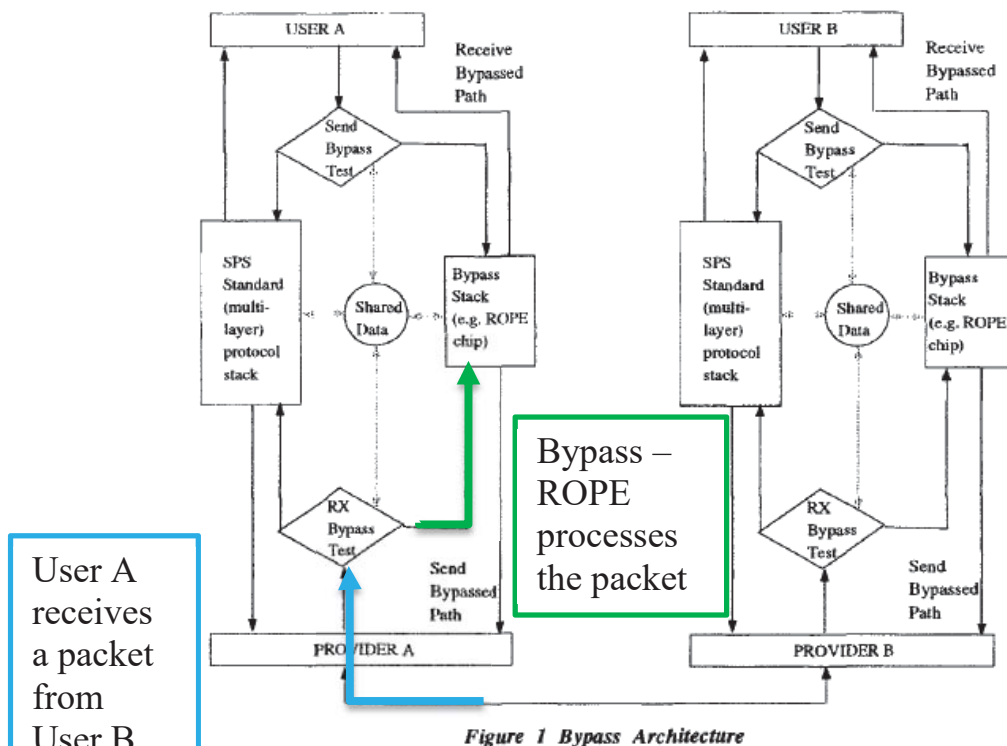


Figure 1 Bypass Architecture

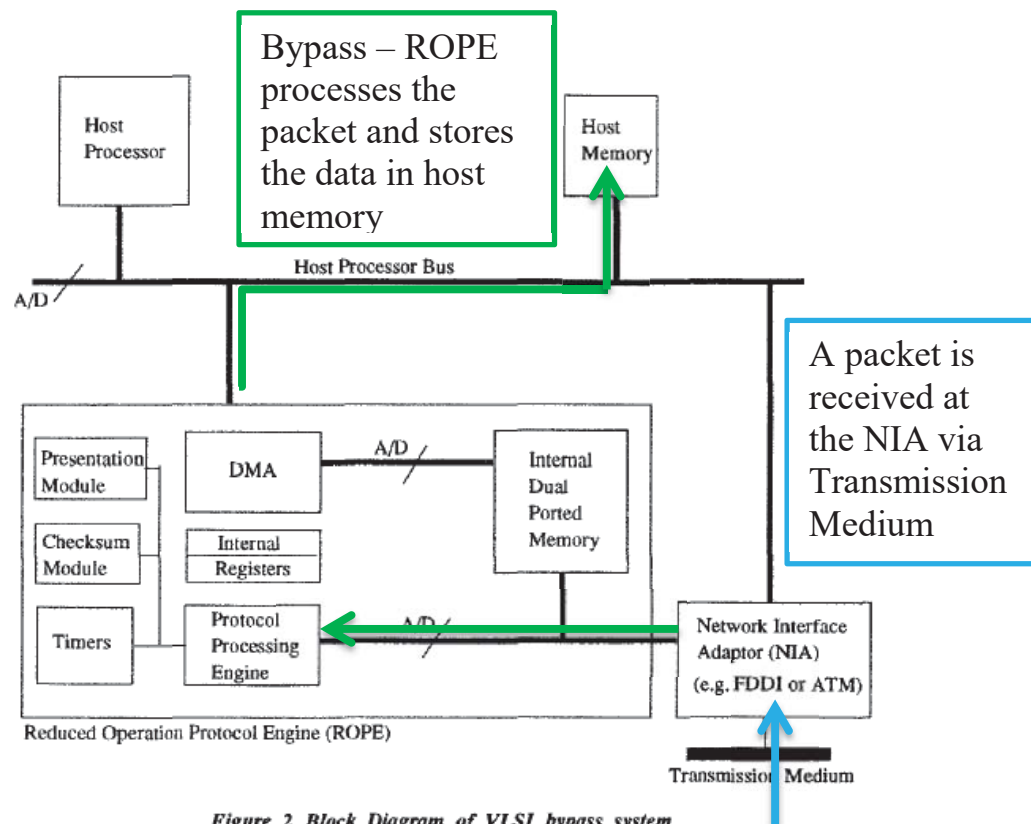
Ex.1015, Thia at .003, Fig.1.

As also explained above for claim limitation [1.7], Thia in view of Tanenbaum96 in further view of 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*). I have illustrated this in Figure 2 of Thia below:

APPENDIX A
to Petition for IPR of U.S. 8,805,948

Thia in view of Tanenbaum96 in further view of Stevens2

[17.2.6] [the network interface having logic that] 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.



Ex.1015, Thia at .007, Fig.2.

As also explained above for claim limitation [1.7], 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 further explained for claim limitation [1.7], 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).

APPENDIX A
to Petition for IPR of U.S. 8,805,948

Thia in view of Tanenbaum96 in further view of Stevens2

[17.2.6] [the network interface having logic that] 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.

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*).

APPENDIX A
to Petition for IPR of U.S. 8,805,948

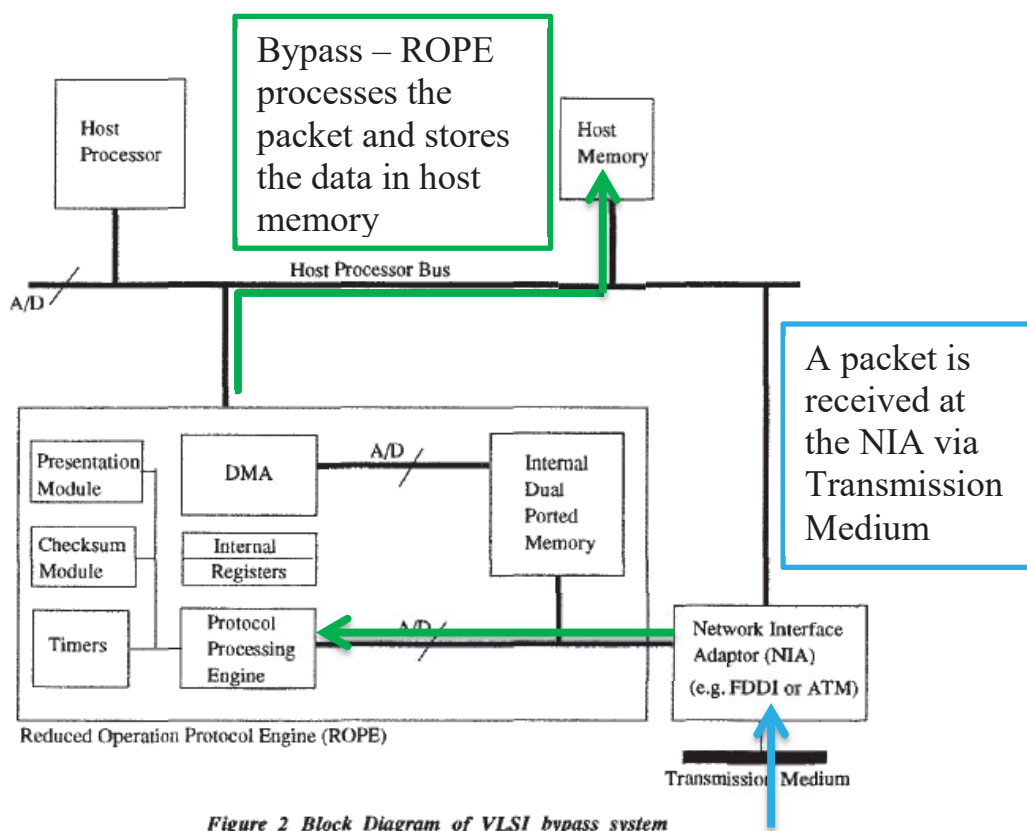
Thia in view of Tanenbaum96 in further view of Stevens2

[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 view of Tanenbaum96 in further view of Stevens2 discloses 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.

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.

As explained for claim limitation [1.7], Thia teaches the use of DMA to move packet data from the ROPE chip to host memory (a host buffer). I've illustrated this in Figure 2 of Thia below:



APPENDIX A
to Petition for IPR of U.S. 8,805,948

Thia in view of Tanenbaum96 in further view of Stevens2
[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.
Ex.1015, Thia at .007, Fig.2.

APPENDIX A
to Petition for IPR of U.S. 8,805,948

Thia in view of Tanenbaum96 in further view of Stevens2
[21] The apparatus of claim 17, wherein the exception conditions include having a RST flag set.
Thia in view of Tanenbaum96 in further view of Stevens2 discloses the apparatus of claim 17, 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 for claim limitation [1.3] disclosing an exception condition including a RST flag set for claim 7 also disclose the limitations of this claim.

APPENDIX A
to Petition for IPR of U.S. 8,805,948

Thia in view of Tanenbaum96 in further view of Stevens2
[22] The apparatus of claim 17, wherein the exception conditions include having a SYN flag set.
Thia in view of Tanenbaum96 in view of Stevens2 discloses the apparatus of claim 17, 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 for claim limitation [1.4] showing an exception condition that includes having a SYN flag set meeting clam 8 also disclose the limitations of this claim.