

U.S. Patent No. 8,131,880 (880 Patent)

IPR2017-01409 (Intel)
IPR2017-01736 (Cavium)
IPR2018-00338 (Dell)
IPR2017-01410 (Intel)
IPR2017-1737 (Cavium)
IPR2018-0339 (Dell)

*All citations herein are to the IPR2017-01391 case unless otherwise noted.

880 Patent: Instituted Grounds

- 2017IPR-01409, IPR2017-01736, IPR2018-00338
 - Ground 1: Thia (Ex. 1015) in view of **Tanenbaum96** (Ex.1006)
 - Claims **1**, 5-10, 12, 14, 16, 17, 20-23, 27, 28, 45, and 55
- 2017IPR-01410, IPR2017-01737, IPR2018-00339
 - Ground 1: Thia (Ex. 1015) in view of **Tanenbaum96** (Ex. 1006)
 - Claims **32**, 34, 35, 39, **41**, 42, and **43**
 - Ground 2: Thia (Ex. 1015) in view of **Tanenbaum96** (Ex. 1006) and **Nahum** (Ex. 1079)
 - Claims 37 and 38

Ex. 1006 – Tanenbaum, Andrew S., Computer Networks (“Tanenbaum96”)

Ex. 1015 – Tia, Y.H., Woodside, C.M. Publication (“Thia”)

Ex. 1079 – Nahum, Erich, Professional Issues in Parallelized Network Protocols (“Nahum”)

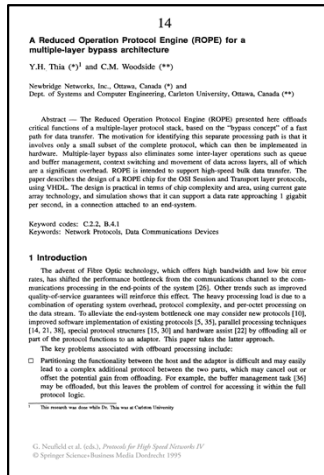
880 Patent: Disputes

1. A POSA would have combined Thia and Tanenbaum96 (and Nahum)
2. Thia and Nahum are enabling
3. The prior art combinations disclose the limitations of the challenged claims of the 880 Patent
4. Motions to Amend 880 Patent should be denied

880 Patent: Disputes

1. A POSA would have combined Thia and Tanenbaum96 (and Nahum)
 - a. **A POSA would have understood that Thia's teachings are applicable to TCP/IP**
 - b. The trend towards TCP/IP in the 1990s would motivate combining Thia's bypass architecture with TCP/IP
 - c. Tanenbaum96 does not teach away from the combination
 - d. It would have been obvious to combine Nahum with Thia and Tanenbaum96

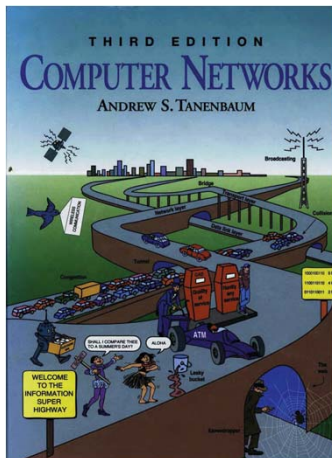
Both disclose a bypass/fast-path based on TCP/IP header prediction



This paper presents a feasibility study for a new approach to hardware assistance. It combines the relatively simple operations needed for data transfer across multiple layers and provides a hardware "fast path" for them, which will be efficient for bulk data transfer. It is based on the "protocol bypass concept" [37] which is a generalization of Jacobson's "Header Prediction" algorithm [20] for TCP/IP. Bypass solves the problems identified above, which may limit the use of offboard processing, by implementing an entire service through all layers for certain cases. This simplifies the interface between the host and the adaptor chip and minimizes their interaction, which is supported by an access test, some DMA processing and a simple command protocol. 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;

See also Paper 1 (1409 Petition) at 32-31; Paper 1 (1410 Petition) at 33-34, 36-37.



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. When this optimization and all the other ones discussed in this chapter are used together, it is possible to get TCP to run at 90 percent of the speed of a local memory-to-memory copy, assuming the network itself is fast enough.

Ex. 1006.585 (Tanenbaum96);

See also Paper 1 (1409 Petition) at 32-31; Paper 1 (1410 Petition) at 33-34, 36-37.

Thia's teachings are not limited to OSI

14

A Reduced Operation Protocol Engine (ROPE) for a multiple-layer bypass architecture

Y.H. Thia (*)¹ and C.M. Woodside (**)

Newbridge Networks, Inc., Ottawa, Canada (*) and
Dept. of Systems and Computer Engineering, Carleton University, Ott

Abstract — The Reduced Operation Protocol Engine (ROPE) presents critical functions of a multiple-layer protocol stack, based on the “bypass concept” of a fast path for data transfer. The motivation for identifying this separate processing path is that it involves only a small subset of the complete protocol, which can then be implemented in hardware. Multiple-layer bypass also eliminates some inter-layer operations such as queue and buffer management, context switching and movement of data across layers, all of which are a significant overhead. ROPE is intended to support high-speed bulk data transfer. The paper describes the design of a ROPE chip for the OSI Session and Transport layer protocols, using VHDL. The design is practical in terms of chip complexity and area, using current gate array technology, and simulation shows that it can support a data rate approaching 1 gigabit per second, in a connection attached to an end-system.

Keyword codes: C.2.2, B.4.1
Keywords: Network Protocols, Data Communications Devices

1 Introduction

The advent of Fibre Optic technology, which offers high bandwidths, has shifted the performance bottleneck from the communication processing in the end-points of the system [26]. Other technologies such as ATM and quality-of-service guarantees will reinforce this effect. The heavy processing overhead, protocol complexity, and protocol overhead in the data stream. To alleviate the end-system bottleneck one may consider improved software implementation of existing protocols [5, 35], parallel processing [14, 21, 38], special protocol structures [15, 30] and hardware assist [2] part of the protocol functions to an adaptor. This paper takes the latter approach.

The key problems associated with offboard processing include:

- Partitioning the functionality between the host and the adaptor is difficult. It may lead to a complex additional protocol between the two parts, which may offset the potential gain from offloading. For example, the buffer management may be offloaded, but this leaves the problem of control for access to the protocol logic.

¹ This research was done while Dr. Thia was at Carleton University

A Reduced Operation Protocol Engine (ROPE) for a multiple-layer bypass architecture

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. The motivation for identifying this separate processing path is that it involves only a small subset of the complete protocol, which can then be implemented in hardware. Multiple-layer bypass also eliminates some inter-layer operations such as queue and buffer management, context switching and movement of data across layers, all of which are a significant overhead. ROPE is intended to support high-speed bulk data transfer. The paper describes the design of a ROPE chip for the OSI Session and Transport layer protocols, using VHDL. The design is practical in terms of chip complexity and area, using current gate array technology, and simulation shows that it can support a data rate approaching 1 gigabit per second, in a connection attached to an end-system.

Ex. 1015.001 (Thia); See also Paper 1 (1409 Petition) at 24-25, Paper 42 (1409 Reply) at 9; Ex. 1223.016-.017 (1409 Lin Reply Decl.) at ¶ 26; Paper 1 (1410 Petition) at 25; Paper 42 (1410 Reply) at 7-8; Ex. 1223.026-.027 (1410 Lin Reply Decl.) at ¶ 37.

Thia's standard protocol stack (SPS) is a "multi-layer" stack, not an "OSI" stack

14

A Reduced Operation Protocol Engine (ROPE) for a multiple-layer bypass arch

Y.H. Thia (*)¹ and C.M. Woo

Newbridge Networks, Inc., Ottawa, Canada (*) and
Dept. of Systems and Computer Engineering, Carleton University, Ottawa, Canada (**)

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. The motivation for identifying this separate processing path is that it involves only a small subset of the complete protocol, which can then be implemented in hardware. Multiple-layer bypass also eliminates some inter-layer operations such as queue and buffer management, context switching and movement of data across layers, all of which are a significant overhead. ROPE is intended to support high-speed bulk data transfer. The paper describes the design of a ROPE chip for the OSI Session and Transport layer protocols, using VHDL. The design is practical in terms of chip complexity and area, using current gate array technology, and simulation shows that it can support a data rate approaching 1 gigabit per second, in a connection attached to an end-system.

Keyword codes: C.2.2, B.4.1
Keywords: Network Protocols, Data Communications Devices

1 Introduction

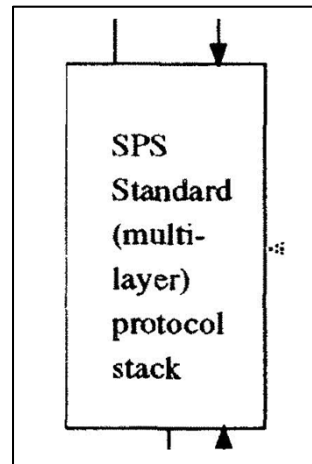
The advent of Fibre Optic technology, which offers high bandwidth and low bit error rates, has shifted the performance bottleneck from the communications channel to the communications processing in the end-points of the system [26]. Other trends such as improved quality-of-service guarantees will reinforce this effect. The heavy processing load is due to a combination of operating system overhead, protocol complexity, and per-octet processing on the data stream. To alleviate the end-system bottleneck one may consider new protocols [10], improved software implementation of existing protocols [5, 35], parallel processing techniques [14, 21, 38], special protocol structures [15, 30] and hardware assist [22] by offloading all or part of the protocol functions to an adaptor. This paper takes the latter approach.

The key problems associated with offboard processing include:

- Partitioning the functionality between the adaptor and the end-system may lead to a complex additional processing path that may offset the potential gain from offloading, but this leaves the protocol logic.

¹ This research was done while Dr. Thia was at C

Figure 1 illustrates the architecture of a bypass implementation for any standard protocol.



without the bypass. The SPS may refer to a single layer or to multiple adjoining layers of a layered protocol stack. The bypass has 4 key components:

Ex. 1015.003 (Thia); See also Paper 1 (1409 Petition) at 25, 30, 34, 35; Paper 42 (1409 Reply) at 9-11; Paper 1 (1410 Petition) at 26, 31-32, 35, 40; Paper 42 (1410 Reply) at 7-9.

Thia teaches that its bypass offload is for more than one multi-layer stack

14

A Reduced Operation Protocol Engine (ROPE) for a multiple-layer bypass architecture

Y.H. Thia (*)¹ and C.M. Woodside (**)

Newbridge Networks, Inc., Ottawa, Canada (*) and
Dept. of Systems and Computer Engineering, Carleton University, Ottawa, Canada (**)

Abstract — The Reduced Operation Protocol Engine (ROPE) provides critical functions of a multiple-layer protocol stack, based on the “bypass” path for data transfer. The motivation for identifying this separate protocol involves only a small subset of the complete protocol, which can be implemented in hardware. Multiple-layer bypass also eliminates some inter-layer operations and buffer management, context switching and movement of data across layers, which are a significant overhead. ROPE is intended to support high-speed bypass processing. This paper describes the design of a ROPE chip for the OSI Session and Transport layers using VHDL. The design is practical in terms of chip complexity and array technology, and simulation shows that it can support a data rate of 100 Mbit per second, in a connection attached to an end-system.

Keyword codes: C.2.2, B.4.1
Keywords: Network Protocols, Data Communications Devices

1 Introduction

The advent of Fibre Optic technology, which offers high bandwidths, has shifted the performance bottleneck from the communication links to communications processing in the end-points of the system [26]. Other technologies and quality-of-service guarantees will reinforce this effect. The heavy processing and combination of operating system overhead, protocol complexity, and processing of the data stream. To alleviate the end-system bottleneck one may consider an improved software implementation of existing protocols [5, 35], parallel processing techniques [14, 21, 38], special protocol structures [15, 30] and hardware assist [22] by offloading all or part of the protocol functions to an adaptor. This paper takes the latter approach.

The key problems associated with offboard processing include:

- Partitioning the functionality between the host and the adaptor is difficult and may easily lead to a complex additional protocol between the two parts, which may cancel out or offset the potential gain from offloading. For example, the buffer management task [36] may be offloaded, but this leaves the problem of control for accessing it within the full protocol logic.

¹ This research was done while Dr. Thia was at Carleton University

- A clean separation of functionality requiring only a simple protocol to communicate between the host and adaptor is desired, and is provided by a bypass. Its particular set of functions are complete in themselves and have a focussed interface with the host software at the packet entry point. There is relatively infrequent switching between the SPS and the bypass stack;
- Reduced non-protocol-specific processing overhead. For example the processing of acknowledgment packets is dominated by interrupt handling, typically a few hundred instructions, rather than by the protocol processing itself. Our approach removes acknowledgment handling altogether from the host. Also, the bypass system can be extended to incorporate multiple-layer stacks and remove overhead that way;

Ex. 1015.005 (Thia); See also Paper 42 (1409 Reply) at 9;
Paper 42 (1410 Reply) at 7.

TCP/IP and OSI were widely understood to be very similar

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

INTEL CORPORATION
Petitioner

v.

ALACRITECH, INC.
Patent Owner

Case IPR. No. Unassigned
U.S. Patent No. 8,131,880
Title: INTELLIGENT NETWORK INTERFACE DEVICE AND SYSTEM FOR
ACCELERATED COMMUNICATION

DECLARATION OF BILL LIN IN SUPPORT OF PETITION
FOR *INTER PARTES* REVIEW OF
U.S. PATENT NO. 8,131,880
UNDER 37 C.F.R. § 1.68

Mail Stop "PATENT BOARD"
Patent Trial and Appeal Board
U.S. Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450

INTEL Ex.1003.001

Contrasting the OSI and the TCP/IP Models

The OSI Model

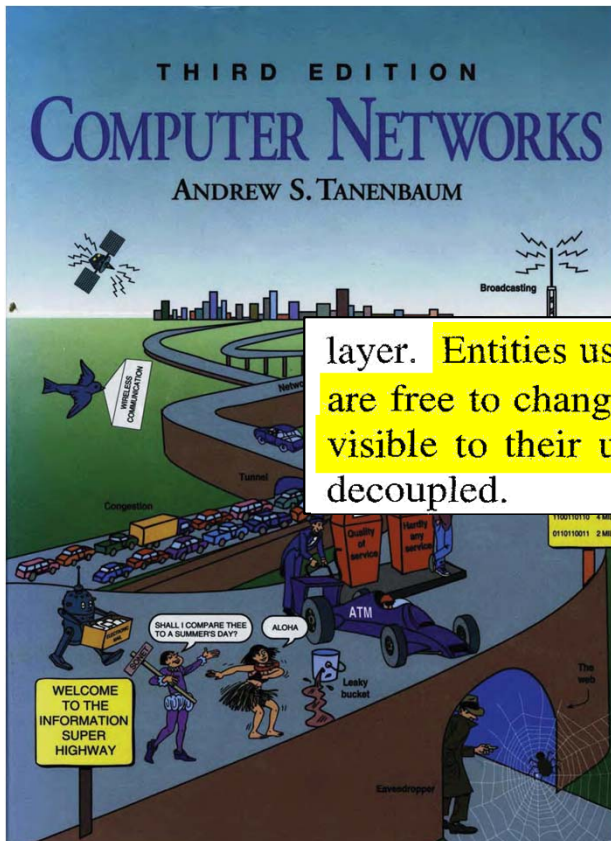
- Application Layer
- Presentation Layer
- Session Layer
- Tranport Layer
- Network Layer
- Data Layer
- Physical Layer

The TCP/IP Model

- Application layer
- Tranport layer
- Internet layer
- Network Access Layer

Ex. 1003.011 (IPR2017-1409 Lin Decl.); *See also* Paper 1 (1409 Petition) at 30-35; Ex.1003.068-.074 (1409 Lin Decl.); Paper 1 (1410 Petition) at 31-40; Ex.1003.069-.080 (1410 Lin Decl.).

Layered protocols mean TCP/IP can be substituted for OSI



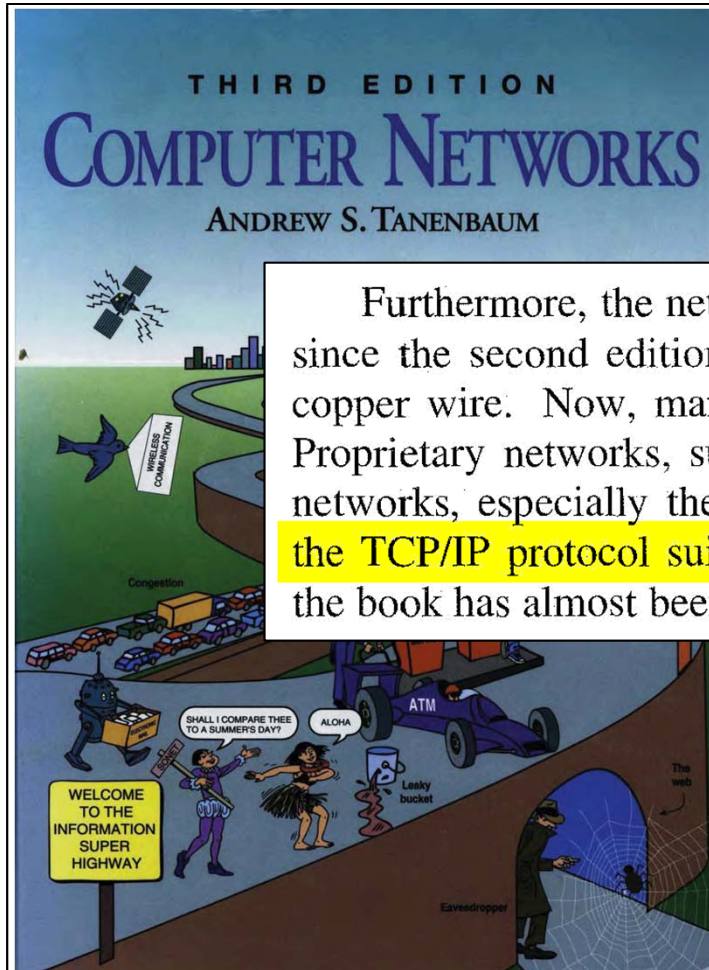
layer. Entities use protocols in order to implement their service definitions. They are free to change their protocols at will, provided they do not change the service visible to their users. In this way, the service and the protocol are completely decoupled.

Ex. 1006.045-.046 (Tanenbaum96);
See also Paper 1 (1409 Petition) at 16, 34-35;
Paper 1 (1410 Petition) at 16-17, 35-36, 39-40.

880 Patent: Disputes

1. A POSA would have combined Thia and Tanenbaum96 (and Nahum)
 - a. A POSA would have understood that Thia's teachings are applicable to TCP/IP
 - b. The trend towards TCP/IP in the 1990s would motivate combining Thia's bypass architecture with TCP/IP**
 - c. Tanenbaum96 does not teach away from the combination
 - d. It would have been obvious to combine Nahum with Thia and Tanenbaum96

By 1996 OSI protocol use vanished and TCP/IP became dominant



Furthermore, the networking hardware and software have completely changed since the second edition appeared. In 1988, nearly all networks were based on copper wire. Now, many are based on fiber optics or wireless communication. Proprietary networks, such as SNA, have become far less important than public networks, especially the Internet. **The OSI protocols have quietly vanished, and the TCP/IP protocol suite has become dominant.** In fact, so much has changed, the book has almost been rewritten from scratch.

Ex. 1006.016 (Tanenbaum96);
See also Paper 1 (1409 Petition) at 28, 32-33; Paper 1 (1410 Petition) at 28, 34-35.

Thia's hardware offload provides advantages over software alone

14

A Reduced Operation Protocol Engine (ROPE) for a multiple-layer bypass architecture

Y.H. Thia (*)¹ and C.M. Woodside (**)

It can be concluded from this study that it is feasible to implement the bypass stack (at least for the transport and session layers) in VLSI and that the performance would be at least an order of magnitude higher than software protocol processing. The bypass system offloads the critical protocol functions and the associated non-protocol-specific functions onto a "Reduced Operation Protocol Engine" (ROPE). The gate count for the bypass chip can easily fit into a commercially available gate array Integrated Circuit. Per-octet operations are particularly efficient when performed on the chip. The host processor is relieved of a significant proportion of protocol processing and can concentrate on the application processing. The speed of communication processing in the host system can now match the transmission bandwidth of high-speed networks, e.g. ATM technology, thereby increasing the application-to-application throughput performance. (In an ATM system we assume that the segmentation

lead to a complex additional protocol between the two parts, which may cancel out or offset the potential gain from offloading. For example, the buffer management task [36] may be offloaded, but this leaves the problem of control for accessing it within the full protocol logic.

¹ This research was done while Dr. Thia was at Carleton University

Ex. 1015.013 (Thia);
See also Paper 1 (1409 Petition) at 33-34, 41;
Ex.1003.060, .072-.073 (1409 Lin Decl.);
Paper 1 (1410 Petition) 34-35, 37-38, 61-62;
Ex.1003.059-.060, .072-.074, .077-.078 (1410 Lin Decl.).

880 Patent: Disputes

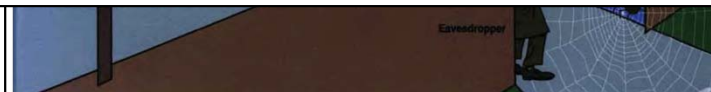
1. A POSA would have combined Thia and Tanenbaum96 (and Nahum)
 - a. A POSA would have understood that Thia's teachings are applicable to TCP/IP
 - b. The trend towards TCP/IP in the 1990s would motivate combining Thia's bypass architecture with TCP/IP
 - c. **Tanenbaum96 does not teach away from the combination**
 - d. It would have been obvious to combine Nahum with Thia and Tanenbaum96

Tanenbaum96 does not teach away from a combination with Thia



Instead, it describes design preferences and tradeoffs

A tempting way to go fast is to build fast network interfaces in hardware. The difficulty with this strategy is that unless the protocol is exceedingly simple, hardware just means a plug-in board with a second CPU and its own program. **To avoid having the network coprocessor be as expensive as the main CPU, it is often a slower chip.** The consequence of this design is that much of the time the main (fast) CPU is idle waiting for the second (slow) CPU to do the critical work. It is a myth to think that the main CPU has other work to do while waiting. Furthermore, when two general-purpose CPUs communicate, race conditions can occur, so elaborate protocols are needed between the two processors to synchronize them correctly. **Usually, the best approach is to make the protocols simple and have the main CPU do the work.**



See also Paper 42 (1409 Reply) at 7-8; Ex. 1223.013-.016 (1409 Lin Reply Decl.); Paper 42 (1410 Reply) at 5-6; Ex. 1223.023-.025 (1410 Lin Reply Decl.).

Tanenbaum96 does not discourage offloading simple protocols

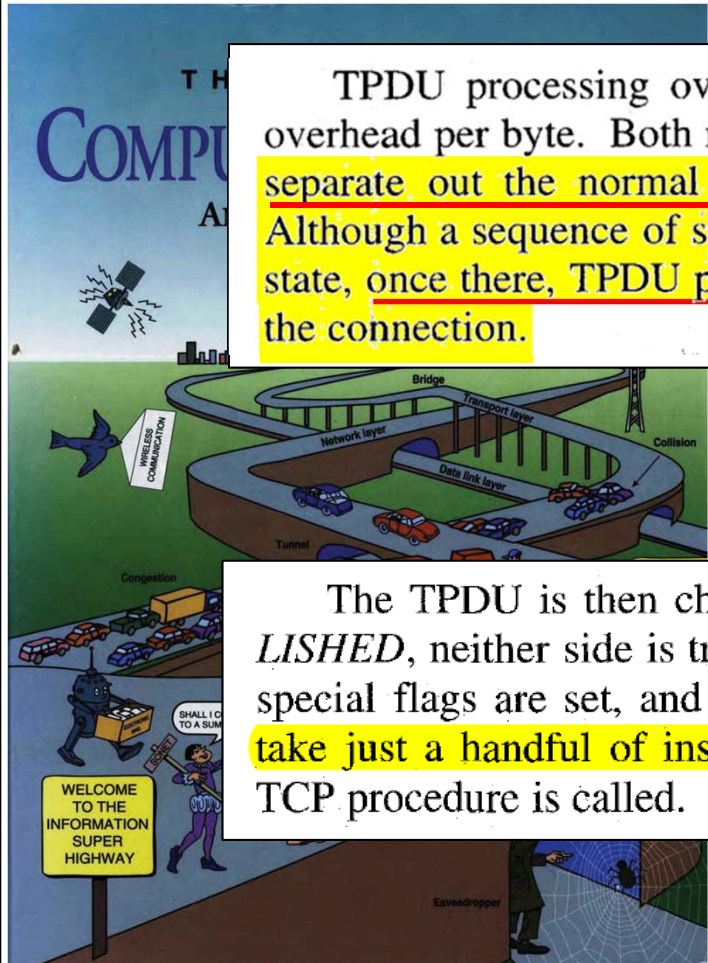
THIRD EDITION
COMPUTER NETWORKS
ANDREW S. TANENBAUM

A tempting way to go fast is to build fast network interfaces in hardware. The difficulty with this strategy is that unless the protocol is exceedingly simple, hardware just means a plug-in board with a second CPU and its own program. To avoid having the network coprocessor be as expensive as the main CPU, it is often a slower chip. The consequence of this design is that much of the time the main



Ex. 1006.588 (Tanenbaum96);
See also Paper 42 (1409 Reply) at 7; Ex. 1223.014-.015 (1409 Lin Reply Decl.);
Paper 42 (1410 Reply) at 5-6; Ex. 1223.024-.025 (1410 Lin Reply Decl.).

Tanenbaum96: Transport processing is “straightforward” in the “normal case”



TPDU processing overhead has two components: overhead per TPDU and overhead per byte. Both must be attacked. 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's are needed to get into the *ESTABLISHED* state, once there, TPDU processing is straightforward until one side starts to close the connection.

Ex. 1006.583 (Tanenbaum96);

See also Paper 42 (1409 Reply) at 7; Ex. 1223.014-.015 (1409 Lin Reply Decl.);
Paper 42 (1410 Reply) at 5-6; Ex. 1223.024-.025 (1410 Lin Reply Decl.).

The TPDU 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.

Ex. 1006.585 (Tanenbaum96);

See also Paper 42 (1409 Reply) at 7; Ex. 1223.014-.015 (1409 Lin Reply Decl.);
Paper 42 (1410 Reply) at 5-6; Ex. 1223.024-.025 (1410 Lin Reply Decl.).

Thia also recognizes the difficulty of offloading a complex protocol stack

14

A Reduced Operation Protocol Engine (ROPE) for a multiple-layer bypass architecture

Y.H. Thia (*)¹ and C.M. Woodside (**)

Newbridge Networks, Inc., Ottawa, Canada (*) and
Dept. of Systems and Computer Engineering, Carleton University, Ottawa, Canada (**)

Abstract — The Reduced Operation Protocol Engine (ROPE) presented here offloads critical functions of a multiple-layer protocol stack based on the “busess concept” of a fast

- The choice of hardware for the adaptor depends on the complexity of the functions it supports. In [2, 22] where the transport protocol layer is offloaded or in [7] where the full protocol stack can be offloaded, general purpose microprocessors are used. Probably because of the complexity of existing protocols, VLSI [24] implementation above the data link layer has been disappointing so far. In [8], dedicated VLSI chips are used to support TCP checksums. Also, some newer lightweight transport protocols are specially designed for VLSI implementation [1, 3].

improved software implementation of existing protocols [5, 35], parallel processing techniques [14, 21, 38], special protocol structures [15, 30] and hardware assist [22] by offloading all or part of the protocol functions to an adaptor. This paper takes the latter approach.

The key problems associated with offboard processing include:

- Partitioning the functionality between the host and the adaptor is difficult and may easily lead to a complex additional protocol between the two parts, which may cancel out or offset the potential gain from offloading. For example, the buffer management task [36] may be offloaded, but this leaves the problem of control for accessing it within the full protocol logic.

¹ This research was done while Dr. Thia was at Carleton University

Ex. 1015.002 (Thia);
See also Paper 42 (1409 Reply) at 8;
Ex. 1223.015-.016 (1409 Lin Reply Decl.); Paper 42 (1410 Reply) at 6;
Ex. 1223.025 (1410 Lin Reply Decl.).

Thia's solution: "Fast path" offload based on header prediction

14

A Reduced Operation Protocol Engine (ROPE) for a multiple-layer bypass architecture

Y.H. Thia (*)¹ and C.M. Woodside (**)

Newbridge Networks, Inc., Ottawa, Canada (*) and
Dept. of Systems and Computer Engineering, Carleton University, Ottawa, Canada (**)

Abstract — The Reduced Operation Protocol Engine (ROPE) presented here offloads critical functions of a multiple layer protocol stack based on the "bypass concept" of [37].

This paper presents a feasibility study for a new approach to hardware assistance. It combines the relatively simple operations needed for data transfer across multiple layers and provides a hardware "fast path" for them, which will be efficient for bulk data transfer. It is based on the "protocol bypass concept" [37] which is a generalization of Jacobson's "Header Prediction" algorithm [20] for TCP/IP. Bypass solves the problems identified above, which may limit the use of offboard processing, by implementing an entire service through all layers for certain cases. This simplifies the interface between the host and the adaptor chip

improved software implementation of existing protocols [5, 35], parallel processing techniques [14, 21, 38], special protocol structures [15, 30] and hardware assist [22] by offloading all or part of the protocol functions to an adaptor. This paper takes the latter approach.

The key problems associated with offboard processing include:

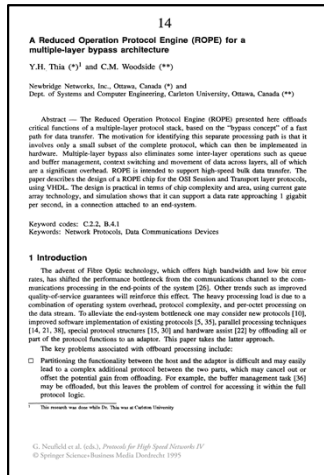
- Partitioning the functionality between the host and the adaptor is difficult and may easily lead to a complex additional protocol between the two parts, which may cancel out or offset the potential gain from offloading. For example, the buffer management task [36] may be offloaded, but this leaves the problem of control for accessing it within the full protocol logic.

¹ This research was done while Dr. Thia was at Carleton University

Ex. 1015.002 (Thia);

See also Paper 42 (1409 Reply) at 8; Ex. 1223.015-.016 (1409 Lin Reply Decl.);
Paper 42 (1410 Reply) at 6; Ex. 1223.025 (1410 Lin Reply Decl.).

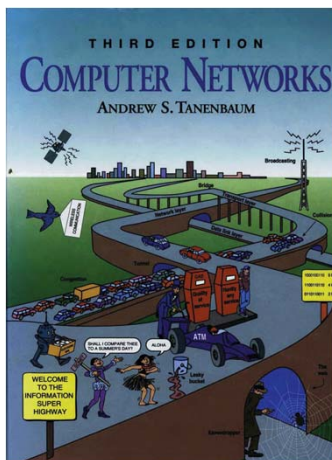
Both disclose a bypass/fast-path based on TCP/IP header prediction



This paper presents a feasibility study for a new approach to hardware assistance. It combines the relatively simple operations needed for data transfer across multiple layers and provides a hardware “fast path” for them, which will be efficient for bulk data transfer. It is based on the “protocol bypass concept” [37] which is a generalization of Jacobson’s “Header Prediction” algorithm [20] for TCP/IP. Bypass solves the problems identified above, which may limit the use of offboard processing, by implementing an entire service through all layers for certain cases. This simplifies the interface between the host and the adaptor chip and minimizes their interaction, which is supported by an access test, some DMA processing and a simple command protocol. 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;

See also Paper 1 (1409 Petition) at 32-31; Paper 1 (1410 Petition) at 33-34, 36-37.



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. When this optimization and all the other ones discussed in this chapter are used together, it is possible to get TCP to run at 90 percent of the speed of a local memory-to-memory copy, assuming the network itself is fast enough.

Ex. 1006.585 (Tanenbaum96);

See also Paper 1 (1409 Petition) at 32-31; Paper 1 (1410 Petition) at 33-34, 36-37.

880 Patent: Disputes

1. A POSA would have combined Thia and Tanenbaum96 (and Nahum)
 - a. A POSA would have understood that Thia's teachings are applicable to TCP/IP
 - b. The trend towards TCP/IP in the 1990s would motivate combining Thia's bypass architecture with TCP/IP
 - c. Tanenbaum96 does not teach away from the combination
 - d. **It would have been obvious to combine Nahum with Thia and Tanenbaum96**

PO makes no additional arguments regarding a combination with Nahum

processing in a multi-processor host computer system. (Ex. 1079.001.) Nahum does not cure the aforementioned deficiencies in the purported motivation to combine Thia and Tanenbaum, nor have Petitioners alleged that it does. Accordingly, based

Paper 32 (1410 Corrected Response) at 57-58 ;
See also Paper 42 (1410 Reply) at 5.

880 Patent: Disputes

1. A POSA would have combined Thia and Tanenbaum96 (and Nahum)
2. Thia and Nahum are enabling
3. The prior art combinations disclose the limitations of the challenged claims of the 880 Patent
4. Motions to Amend 880 Patent should be denied

PO fails to identify why Thia and Nahum are allegedly not enabling

- Patent Owner contends that Thia is an “inoperable device” and is therefore a non-enabling reference

Paper 32 (1409 Corrected Response) at 20-21; Paper 32 (1410 Corrected Response) at 16.

- Patent Owner’s only support that Thia and Nahum are not enabling is Dr. Almeroth’s conclusory declaration

See Paper 42 (1409 Reply) at 3-4; Paper 42 (1410 Reply) at 2-3.

- But a non-enabling reference can be prior art “for all that it teaches”

Id. (citing *Beckman Instruments v. LKB Produkter AB*, 892 F.2d 1547, 1551 (Fed. Cir. 1989)).

This is not a theoretical device

4.3 First Design: Design Steps

Figure 3 shows the steps followed in this study. There were three stages, a behavioural model, a structural or RTL model, and a gate level design. These gave us two kinds of feasibility check, that the logic we specified will execute the protocol within the environment we envisage, and that the design is technically feasible, for instance in a reasonable chip area.

Ex. 1015.008 (Thia).

in the semiconductor industry to design semiconductor chips. A person of ordinary skill in the art (“POSA”) would know that a gate-level design can be fabricated into a chip using well-known software tools and chip fabrication facilities. A

Ex. 1223.004-.005 (1409 Lin Reply Decl.) at ¶ 8; see also Ex. 1223.004-.005 (1410 Lin Reply Decl.) at ¶ 8; See also Paper 42 (1409 Reply) at 3-4; Paper 42 (1410 Reply) at 2-3.

Nahum Is enabling

Appears in "Proceedings of the First Symposium on Operating Systems Design and Implementation," Usenix Association, November 1994.

Performance Issues in Parallelized Network Protocols

Erich M. Nahum, David J. Yates, James F. Kurose, and Don Towsley*

Department of Computer Science
University of Massachusetts
Amherst, MA 01003

Abstract

Parallel processing has been proposed as a means of improving network protocol throughput. Several different strategies have been taken towards parallelizing protocols. A relatively popular approach is *packet-level parallelism*, where packets are distributed across processors.

This paper provides an experimental performance study of packet-level parallelism on a contemporary shared-memory multiprocessor. We examine several unexplored areas in packet-level parallelism and investigate how various protocol structuring and implementation techniques can affect performance. We study TCP/IP and UDP/IP protocol stacks, implemented with a parallel version of the *x*-kernel running in user space on Silicon Graphics multiprocessors.

Our results show that only limited packet-level parallelism can be achieved within a single connection under TCP, but that using multiple connections can improve available parallelism. We also demonstrate that packet ordering plays a key role in determining single-connection TCP performance, that careful use of locks is a necessity, and that selective exploitation of caching can improve throughput. We also describe experiments that compare parallel protocol performance on two generations of a parallel machine and show how computer architectural trends can influence performance.

1 Introduction

Parallel processing has been proposed as a means of improving network protocol throughput. Two trends motivate the use of parallelism in network processing. First, network bandwidths are increasing by orders of magnitude, with the advent of technologies such as ATM. Second, shared-memory multiprocessors are becoming more common, as

*This research supported in part by NSF under grant NCR-9206908 and ARPA under contract number F19628-92-C-0089. Erich Nahum was supported by an ARPA Research Assistantship in Parallel Processing. David Yates is the recipient of a Motorola Codex University Partnership in Research Grant. The authors can be reached at {nahum, yates, kurose, towsleyj}@cs.umass.edu.

shown by recent vendor introductions [1, 8, 9] thus an opportunity to exploit the potential of parallel network protocol processing, and this has become an area of research.

The approach we study here is that of *packet-level parallelism*, sometimes referred to as thread-per-processor-per-message parallelism. Originally, Hutchinson and Peterson in the *x*-kernel [14], it distributes packets across processors, achieved both with multiple connections and within a single connection. Packets can be processed on any processor, maximizing flexibility and utilization. Other system approaches include [5, 11].

Several other approaches to parallelism have been proposed and are briefly described here; more surveys can be found in [5, 11]. In *layered* protocols are assigned to specific processors, messages passed between layers through interprocessor communication. Parallelism gains can be achieved through pipelining effects. An example is for *Connection-level parallelism* associates connections with a single processor or thread, achieving speedup proportional to the number of connections. Multiprocessor STREAMS matches this model [26, 27]. *Functional* decomposes functions within a single processor into processing elements. Examples [19, 23, 25]. The relative merits of one approach over the others depends on many factors, including architecture, the number of connections, where parallelism is in hardware or software, the thread scheduling policies employed, and the cost of primitives such as context switching. Schmidt and Suda [2] packet-level parallelism and connection-level parallelism generally perform better than layer parallelism on a shared-memory multiprocessor, due to the context-switching overhead when crossing layers using layer parallelism.

This paper provides an experimental performance study of packet-level parallelism using TCP/IP and UDP/IP protocol stacks. We have conducted this study in the context of a multiprocessor implementation of the *x*-kernel.

2 Experimental Environment

As stated earlier, our environment is based on a parallelized *x*-kernel, and as such, it is similar in several respects to the platform described by Bjorkman and Gunningberg at the Swedish Institute of Computer Science (SICS) [4, 5]. Our platform was, for the most part, developed independently, and for a different type of machine. The exception is the SICS MP TCP code, which we used to guide the design of our parallel TCP, as described in Section 5.1. The SICS platform, however, was based on the February 1992 release of the *x*-kernel, and ran on the Sequent Symmetry. Our environment is based on the December 1993 *x*-kernel release, and runs on the SGI Challenge. Given the differences in hardware, host operating systems, versions of the *x*-kernel infrastructure and protocols, a direct comparison is thus not possible. Where applicable, however, we describe differences between the systems.

INTEL Ex.1079.001

Ex. 1079.002 (Nahum);
See also Paper 42 (1410 Reply) at 5.

880 Patent: Disputes

1. A POSA would have combined Thia and Tanenbaum96 (and Nahum)
2. Thia and Nahum are enabling
3. **The prior art combinations disclose the limitations of the challenged claims of the 880 Patent**
4. Motions to Amend 880 Patents should be denied

880 Patent: Disputes

3. The prior art combinations disclose the limitations of the challenged claims of the 880 Patent
 - a. **The prior art combination renders obvious “an operation code” (claims 1, 17, 32, 34, 45)**
 - b. This discloses a “re-assembler” on, or “re-assembly” by, a network interface (claims 32, 41, 43)
 - c. A “flow key” that includes a “first hop medium access control (MAC) layer address” would have been obvious (claim 32)
 - d. The prior art combination discloses storing the “header portion in a header buffer” if the “header conforms to the TCP protocol” (claim 32)
 - e. The prior art combination discloses a “processor” for TCP processing (claims 1, 32, 41, 43)

880 Patent: Claims 1, 17, 32, 34, and 45

(12) **United States Patent**
Boucher et al.

(10) Patent No.: US 8,131,880 B2
(45) Date of Patent: *Mar. 6, 2012

1. A method of transferring a packet to a host computer system, wherein the packet is received at a communication device from a network, comprising:

parsing a header portion of a first packet received at a network interface for the host computer system to determine if said first packet conforms to a TCP protocol; generating a flow key to identify a first communication flow that includes said first packet, wherein said flow key includes a TCP connection for the communication flow; associating an **operation code** with said first packet, wherein said **operation code** indicates a status of said first packet, including whether said packet is a candidate for transfer to the host computer system that avoids processing said header portion by the host computer system in accordance with said TCP protocol; and processing, by the network interface, said packet according to the TCP connection, including updating a control block representing the TCP connection on the network interface.



17. The method of claim 1, further comprising storing said **operation code** in a control memory.

32. A method of transferring a packet received at a network interface from a network to a host computer system, comprising:

receiving a packet from a network at a network interface of a host computer system;
parsing a header portion of said packet to extract an identifier of a source entity and an identifier of a destination entity;
generating a flow key from said source identifier and said destination identifier to identify a communication flow comprising said packet, wherein said flow key includes a TCP connection for the communication flow and a first hop medium access control (MAC) layer address;
determining whether a header in said header portion conforms to a pre-selected protocol;
storing said flow key in a database;
associating an **operation code** with said packet, wherein said **operation code** identifies a status of said packet;
storing said packet in a packet memory;
if said header conforms to the TCP protocol:
storing a data portion of said packet in a re-assembly buffer;
storing said header portion in a header buffer; and
processing, by the network interface, said packet according to the TCP connection.

34. The method of claim 32, further comprising storing said **operation code** in a control memory.

45. The method of claim 1, wherein said **operation code** indicates whether the packet corresponds to Transport Control Protocol (TCP).

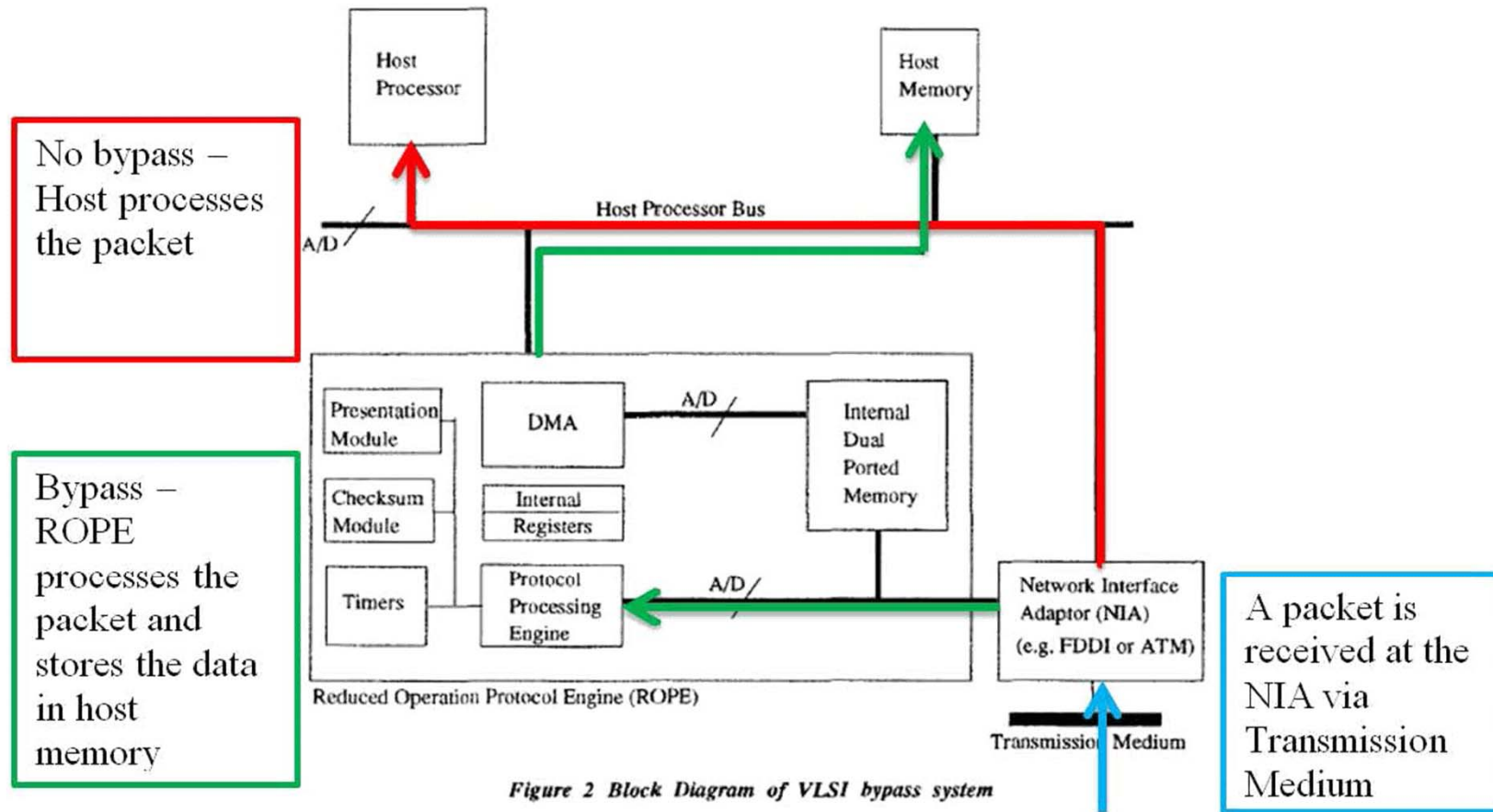
PO told the patent office that a single bit can be an operation code

Applicants: Boucher, et al. Atty. Docket ALA 008F	
conforms to one of a set of communication protocols; and	CCBs or not. This includes all header validation (<i>is it IP, IPV4 or V6, is the IP header checksum correct, is the TCP checksum correct, etc.</i>)
if said header conforms to one of said communication protocols from said identify which s	(6:36-38) "The header is fully parsed by hardware and its type is summarized in a single status word. The checksum is also verified
a flow m store a identify flow;	<p>a flow manager configured to assign an operation code to said packet, wherein said operation code:</p> <ul style="list-style-type: none"> indicates a status of said packet; and indicates a manner of transferring said packet to the host computer system; <p>(85:26-30) "As frames are received, they are placed into 2K-byte DRAM buffers by the Receive hardware sequencer, along with 16 bytes of the above <i>frame status</i>. A <i>pointer</i> to the last byte + 1 of this buffer is queued into the Q_RECV queue. The <i>pointer</i> contains a bit (<i>bit 29</i>) that informs the microcode if this frame is definitely not a fast-path candidate..."</p> <p>(85:38) "<i>If bit 29 is set, this frame is going slow-path.</i>"</p>
a flow m assign a packet, code;	
indicates a status of said packet; and indicates a manner of transferring said packet to the host computer system;	informs the microcode if this frame is definitely not a fast-path candidate..." (85:38) " <i>If bit 29 is set, this frame is going slow-path.</i> "
a packet memory configured to store said packet; and	(14:27) "Receive data buffers are allocated in blocks of 2, 2k bytes each (4k page)."
a transfer module configured to transfer said packet from said packet memory to a host computer system in accordance	(14:1-3) "Data buffers are moved by DMA into the host before the header buffer, since the header buffer contains the status word designating that the data has arrived."
65	

The term "operation code" does not appear in the 880 Patent outside the claims

Ex. 1002 (880 Patent File History) at .249;
See also Paper 1 (1409 Petition) at 49 n.11;
Paper 1 (1410 Petition) at 53 n.10.

Result of the receive bypass test indicates if the packet is bypassable



Ex. 1003.097 (1409 Lin Decl.);
See also Paper 1 (1409 Petition) at 48; Paper 1 (1410 Petition) at 53; Ex. 1003.102 (1410 Lin Decl.).

A POSA would know that the receive bypass test results in an op code

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

with the TCP protocol that avoids processing by the host computer). Persons of ordinary skill in the art would know that a test or check performed produces an output (or result), such as a bit logically indicating true or false, that would then be used by or in other instruction sets or commands. It would have been obvious to indicate the result of the “receive bypass test” using an operation code (*e.g.*, a bit flag, which Patent Owner has argued provides the claimed operation code¹) instructing whether to process the packet on the ROPE chip or on the host. And indeed, Thia teaches that a flag is set signifying that a packet can be fast-pathed:

Title: INTEI

DECLARATION OF BILL LIN IN SUPPORT OF PETITION
FOR *INTER PARTES* REVIEW OF
U.S. PATENT NO. 8,131,880
UNDER 37 C.F.R. § 1.68

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

Ex. 1003.098-.099 (1409 Lin Decl.);
See also Paper 1 (1409 Petition) at 49-50;
Paper 1 (1410 Petition) at 53-54; Ex. 1003.103-.104 (1410 Lin Decl.).

Thia's operation code: Flag used by the "no-in-transit PDU" test

14

A Reduced Operation Protocol Engine (ROPE) for a multiple-layer bypass architecture

Y.H. Thia

Newbridge
Dept. of Sy

Abstract
critical func
path for dat
involves on
hardware,
and buffer
are a signif
paper descr
using VHD
array techn
per second.

Keyword c
Keywords:

2.2 Efficient logic for the bypass test

The "no-in-transit PDU" test can often be avoided. At the beginning of data transfer on a new connection, it is automatically satisfied. It holds as long as no packet fails a bypass test, and it is sufficient to maintain a flag to indicate this. Once a packet fails, and goes to the SPS, then a full "no-in-transit PDU" test must be performed for each packet until the test succeeds, after which control can go back to the flag. Token management and synchronization points of

1 Introduction

The advent of Fibre Optic technology, which offers high bandwidth and low bit error rates, has shifted the performance bottleneck from the communications channel to the communications processing in the end-points of the system [26]. Other trends such as improved quality-of-service guarantees will reinforce this effect. The heavy processing load is due to a combination of operating system overhead, protocol complexity, and per-octet processing on the data stream. To alleviate the end-system bottleneck one may consider new protocols [10], improved software implementation of existing protocols [5, 35], parallel processing techniques [14, 21, 38], special protocol structures [15, 30] and hardware assist [22] by offloading all or part of the protocol functions to an adaptor. This paper takes the latter approach.

The key problems associated with offboard processing include:

- Partitioning the functionality between the host and the adaptor is difficult and may easily lead to a complex additional protocol between the two parts, which may cancel out or offset the potential gain from offloading. For example, the buffer management task [36] may be offloaded, but this leaves the problem of control for accessing it within the full protocol logic.

¹ This research was done while Dr. Thia was at Carleton University

G. Neufeld et al. (eds.), *Protocols for High Speed Networks IV*
© Springer Science+Business Media Dordrecht 1995

INTEL Ex.1015.001

- Thia's flag indicates the status of the most recently-received packet – i.e. whether it will be processed on the bypass fast-path

Ex.1015.004 (Thia); See also Paper 1 (1409 Petition) at 49-50; Ex. 1003.098-.099 (1409 Lin Decl.); Paper 42 (1409 Reply) at 15-16; Paper 1 (1410 Petition) at 53-54; Ex. 1003.103-.104 (1410 Lin Decl.); Paper 42 (1410 Reply) at 15.

880 Patent: Disputes

3. The prior art combinations disclose the limitations of the challenged claims of the 880 Patent
 - a. The prior art combination renders obvious “an operation code” (claims 1, 17, 32, 34, 45)
 - b. **This discloses a “re-assembler” on, or “re-assembly” by, a network interface (claims 32, 41, 43)**
 - c. A “flow key” that includes a “first hop medium access control (MAC) layer address” would have been obvious (claim 32)
 - d. The prior art combination discloses storing the “header portion in a header buffer” if the “header conforms to the TCP protocol” (claim 32)
 - e. The prior art combination discloses a “processor” for TCP processing (claims 1, 32, 41, 43)

880 Patent: Claim 32

(12) **United States Patent**
Boucher et al.

(10) Patent No.: **US 8,131,880 B2**
(45) Date of Patent: ***Mar. 6, 2012**

(54) **INTELLIGENT NETWORK INTERFACE DEVICE AND SYSTEM FOR ACCELERATED COMMUNICATION**

(75) Inventors: **Laurence B. Boucher**, Saratoga, CA (US); **Stephen E. J. Blightman**, San Jose, CA (US); **Peter K. Craft**, San Francisco, CA (US); **David A. Higgen**, Saratoga, CA (US); **Clive M. Philbrick**, San Jose, CA (US); **Daryl D. Starr**, Milpitas, CA (US)

(73) Assignee: **Alacritech, Inc.**, San Jose, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 2126 days. This patent is subject to a terminal disclaimer.

(21) Appl. No.: **10/601,237**

(22) Filed: **Jun. 19, 2003**

(65) **Prior Publication Data**
US 2004/0062246 A1 Apr. 1, 2004

Related U.S. Application Data

(63) Continuation of application No. 10/005,536, filed on Nov. 7, 2001, now Pat. No. 7,167,926, and a continuation-in-part of application No. 09/514,425, filed on Feb. 28, 2000, now Pat. No. 6,427,171, and a continuation-in-part of application No. 09/464,283, filed on Dec. 15, 1999, now Pat. No. 6,427,173, and a continuation of application No. 09/384,792, filed on Aug. 27, 1999, now Pat. No. 6,434,620, and a continuation-in-part of application No. 09/067,544, filed on Apr. 27, 1998, now Pat. No. 6,226,680, and a continuation-in-part of application No. 09/141,713, filed on Aug. 28, 1998, now Pat. No. 6,389,479.

(60) Provisional application No. 60/098,296, filed on Aug. 27, 1998, provisional application No. 60/061,809, filed on Oct. 14, 1997.

(51) Int. Cl. **G06F 15/16** (2006.01)

(52) U.S. CL **709/250**

(58) Field of Classification Search **709/238, 709/250**

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,366,538 A 12/1982 Johnson et al. 364/200 (Continued)

FOREIGN PATENT DOCUMENTS

WO WO/98/19412 5/1998 (Continued)

OTHER PUBLICATIONS

U.S. Appl. No. 60/053,240, by Jolitz et al. (listed filing date Jul. 18, 1997).

(Continued)

Primary Examiner — Jerry Dennison
(74) Attorney, Agent, or Firm — Silicon Edge Law Group LLP, Mark A. Lauer

(57) **ABSTRACT**
An intelligent network interface card (NIC) or communication processing device (CPD) works with a host computer for data communication. The device provides a fast-path that avoids protocol processing for most messages, greatly accelerating data transfer and offloading time-intensive processing tasks from the host CPU. The host retains a fallback processing capability for messages that do not fit fast-path criteria, with the device providing assistance such as validation even for slow-path messages, and messages being selected for either fast-path or slow-path processing. A context for a connection is defined that allows the device to move data, free of headers, directly to or from a destination or source in the host. The context can be passed back to the host for message processing by the host. The device contains specialized hardware circuits that are much faster at their specific tasks than a general purpose CPU. A preferred embodiment includes a trio of pipelined processors devoted to transmit, receive and utility processing, providing full duplex communication for four Fast Ethernet nodes.

60 Claims, 40 Drawing Sheets

The diagram shows two protocol stacks. On the left, a host stack (50) includes layers: NETBIOS (66), TCP (65), IP (64), MAC (63), and PHYSICAL (55). On the right, a client stack (52) includes layers: CLIENT (77), TDI (76), TCP (75), IP (74), MAC (73), and PHYSICAL (59). A 'FAST-PATH' (80) is shown as a dashed line connecting the IP layers (64 and 74). A 'SLOW-PATH' (82) is shown as a dashed line connecting the MAC layers (63 and 73). A bidirectional arrow (57) is at the bottom, indicating communication between the physical layers.

Ex. 1001 (880 Patent), Claim 32.

32. A method of transferring a packet received at a network interface from a network to a host computer system, comprising:

- receiving a packet from a network at a network interface of a host computer system;
- parsing a header portion of said packet to extract an identifier of a source entity and an identifier of a destination entity;
- generating a flow key from said source identifier and said destination identifier to identify a communication flow comprising said packet, wherein said flow key includes a TCP connection for the communication flow and a first hop medium access control (MAC) layer address;
- determining whether a header in said header portion conforms to a pre-selected protocol;
- storing said flow key in a database;
- associating an operation code with said packet, wherein said operation code identifies a status of said packet;
- storing said packet in a packet memory;
- if said header conforms to the TCP protocol:
- storing a data portion of said packet in a **re-assembly buffer**;
- storing said header portion in a header buffer; and
- processing, by the network interface, said packet according to the TCP connection.

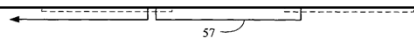
880 Patent: Claims 41 and 43

(12) **United States Patent**
Boucher et al.

(10) Patent No.: US 8,131,880 B2
(45) Date of Patent: *Mar. 6, 2012

41. An apparatus for transferring a packet to a host computer system, comprising:

- a traffic classifier, disposed in a network interface for the host computer system, configured to classify a first packet received from a network by a communication flow that includes said first packet;
- a packet memory, disposed in the network interface, configured to store said first packet;
- a packet batching module, disposed in the network interface, configured to determine whether another packet in said packet memory belongs to said communication flow;
- a flow **re-assembler**, disposed in the network interface, configured to re-assemble a data portion of said first packet with a data portion of a second packet in said communication flow; and
- a processor, disposed in the network interface, that maintains a TCP connection for the communication flow, the TCP connection stored as a control block on the network interface.

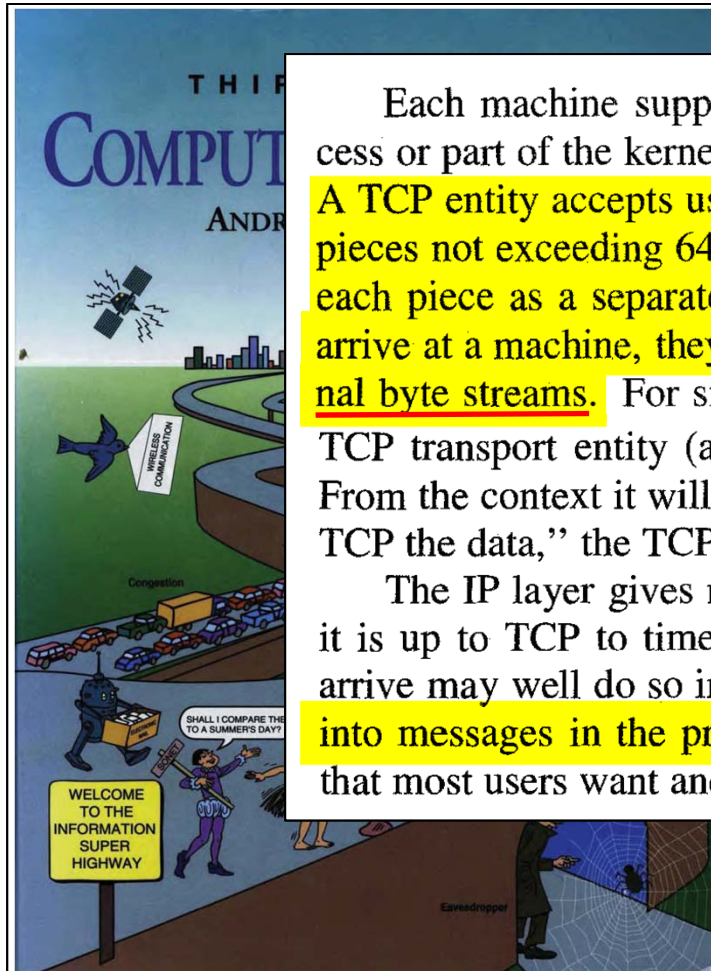


43. A computer system for receiving a packet from a network, comprising:

- a memory configured to store packets received from a network; and
- a network interface for the computer system, the network interface configured to receive a first packet from said network, the network interface comprising:
 - a parser configured to extract information from a header portion of a first packet;
 - a flow manager configured to examine said information;
 - a flow database configured to store an identifier of a first communication flow comprising multiple packets, including said first packet; and
 - a **re-assembler** for storing data portions of said multiple packets without header portions in a first portion of said memory; and
- a processor for processing said first packet and for maintaining a TCP connection for the communication flow, the TCP connection stored as a control block on the network interface.

Ex. 1001 (880 Patent), Claims 41, 43.

Unrebutted evidence that TCP reassembles segments into streams



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. For simplicity, we will sometimes use just “TCP” to mean the TCP transport entity (a piece of software) or the TCP protocol (a set of rules). From the context it will be clear which is meant. For example, in “The user gives TCP the data,” the TCP transport entity is clearly intended.

The IP layer gives no guarantee that datagrams will be delivered properly, so it is up to TCP to time out and retransmit them as need be. Datagrams that do arrive may well do so in the wrong order; it is also up to TCP to reassemble them into messages in the proper sequence. In short, TCP must furnish the reliability that most users want and that IP does not provide.

Ex. 1006.540-.541 (Tanenbaum96);
See also Paper 1 (1410 Petition) at 57-58;
Ex. 1003.109-.110 (1410 Lin Decl.).

Unrebutted evidence of re-assembler / re-assembly in Thia

14

A Reduced Operation Protocol Engine (ROPE) for a multiple-layer bypass architecture

Y.H. Thia (*)¹ and C.M. Woodside (**)

Newbridge Networks, Inc., Ottawa, Canada (*) and
Dept. of Systems and Computer Engineering, Carleton University, Ottawa, Canada (**)

Abstract — The critical functions of a path for data transfer involves only a small hardware. Multiple-layer and buffer management are a significant overhead. This paper describes the design using VHDL. The design uses array technology, and processes data at 100 Mbit per second, in a compact form.

Keyword codes: C.2.2
Keywords: Network Routers

1 Introduction

The advent of Fibre Optic technology, which offers high bandwidth and low bit error rates, has shifted the performance bottleneck from the communications channel to the communications processing in the end-points of the system [26]. Other trends such as improved quality-of-service guarantees will reinforce this effect. The heavy processing load is due to a combination of operating system overhead, protocol complexity, and per-octet processing on the data stream. To alleviate the end-system bottleneck one may consider new protocols [10], improved software implementation of existing protocols [5, 35], parallel processing techniques [14, 21, 38], special protocol structures [15, 30] and hardware assist [22] by offloading all or part of the protocol functions to an adaptor. This paper takes the latter approach.

The key problems associated with offboard processing include:

- Partitioning the functionality between the host and the adaptor is difficult and may easily lead to a complex additional protocol between the two parts, which may cancel out or offset the potential gain from offloading. For example, the buffer management task [36] may be offloaded, but this leaves the problem of control for accessing it within the full protocol logic.

¹ This research was done while Dr. Thia was at Carleton University

G. Neufeld et al. (eds.), *Protocols for High Speed Networks IV*
© Springer Science+Business Media Dordrecht 1995

INTEL Ex.1015.001

transfer. The data portion of a PDU may be physically moved for the following reasons:

- Copying between the adaptor buffer and the host system memory;
- Crossing protection domains (address spaces) — e.g. at the user/kernel boundary. This problem becomes more pronounced in microkernels which treat a protocol task as a server process outside the kernel domain [11];
- Per-octet processing like presentation conversion and checksum routines.

Ex. 1015.005 (Thia); See also Paper 1 (1410 Petition) at 77;
Ex. 1003.138-.139 (1410 Lin Decl.).

“Segmentation/reassembly” refers to lower-layer fragmentation

14

A Reduced Operation Protocol Engine (ROPE) for a multiple-layer bypass architecture

Y.H. Thia (*)¹ and C.M. Woodside (**)

The scope of functions included in a bypass may be narrowly defined, or more extended. A bypass does not include fast connection setup but also does not interfere with it. There is no segmentation/reassembly within the bypass path, but we do not see this as a major restriction, as research suggests that fragmentation of PDUs should be restricted only to the lower layers and should occur only once in the protocol stack [23]. The Segmentation and Reassembly sublayer of the ATM adaptation layer is a good place for such functions [25].

The advent of Fibre Optic technology, which offers high bandwidth and low bit error rates, has shifted the performance bottleneck from the communications channel to the communications processing in the end-points of the system [26]. Other trends such as improved quality-of-service guarantees will reinforce this effect. The heavy processing load is due to a combination of operating system overhead, protocol complexity, and per-octet processing on the data stream. To alleviate the end-system bottleneck one may consider new protocols [10], improved software implementation of existing protocols [5, 35], parallel processing techniques [14, 21, 38], special protocol structures [15, 30] and hardware assist [22] by offloading all or part of the protocol functions to an adaptor. This paper takes the latter approach.

The key problems associated with offboard processing include:

- Partitioning the functionality between the host and the adaptor is difficult and may easily lead to a complex additional protocol between the two parts, which may cancel out or offset the potential gain from offloading. For example, the buffer management task [36] may be offloaded, but this leaves the problem of control for accessing it within the full protocol logic.

¹ This research was done while Dr. Thia was at Carleton University

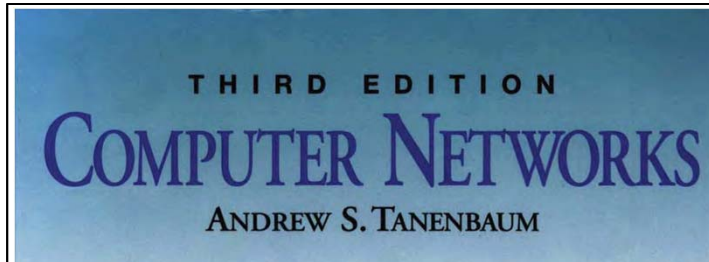
INTEL Ex.1015.001

Ex. 1015.014 (Thia).

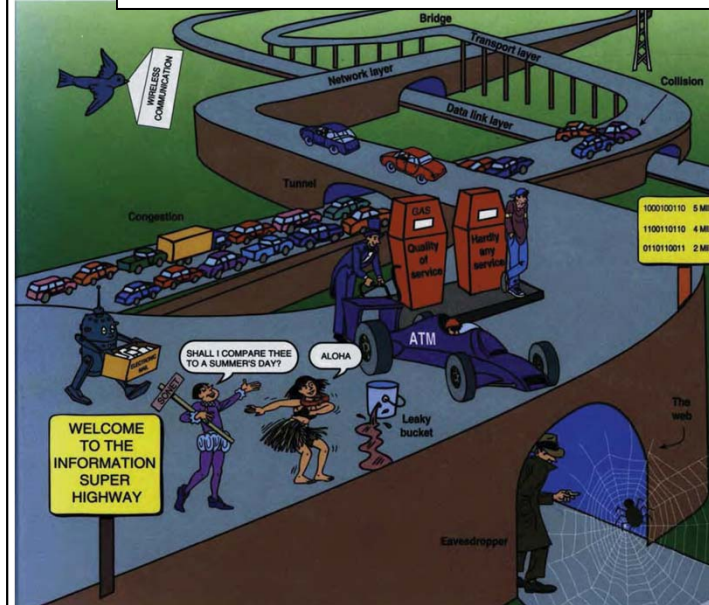
Thia's disclosure is discussing fragmentation and re-assembling those fragments at lower-layer protocols.

See Ex. 1223.017-.020 (1410 Lin Reply) at ¶¶ 24-28; see also Paper 42 (1410 Reply) at 17-18.

Network layer: “Segmentation” is “fragmentation”

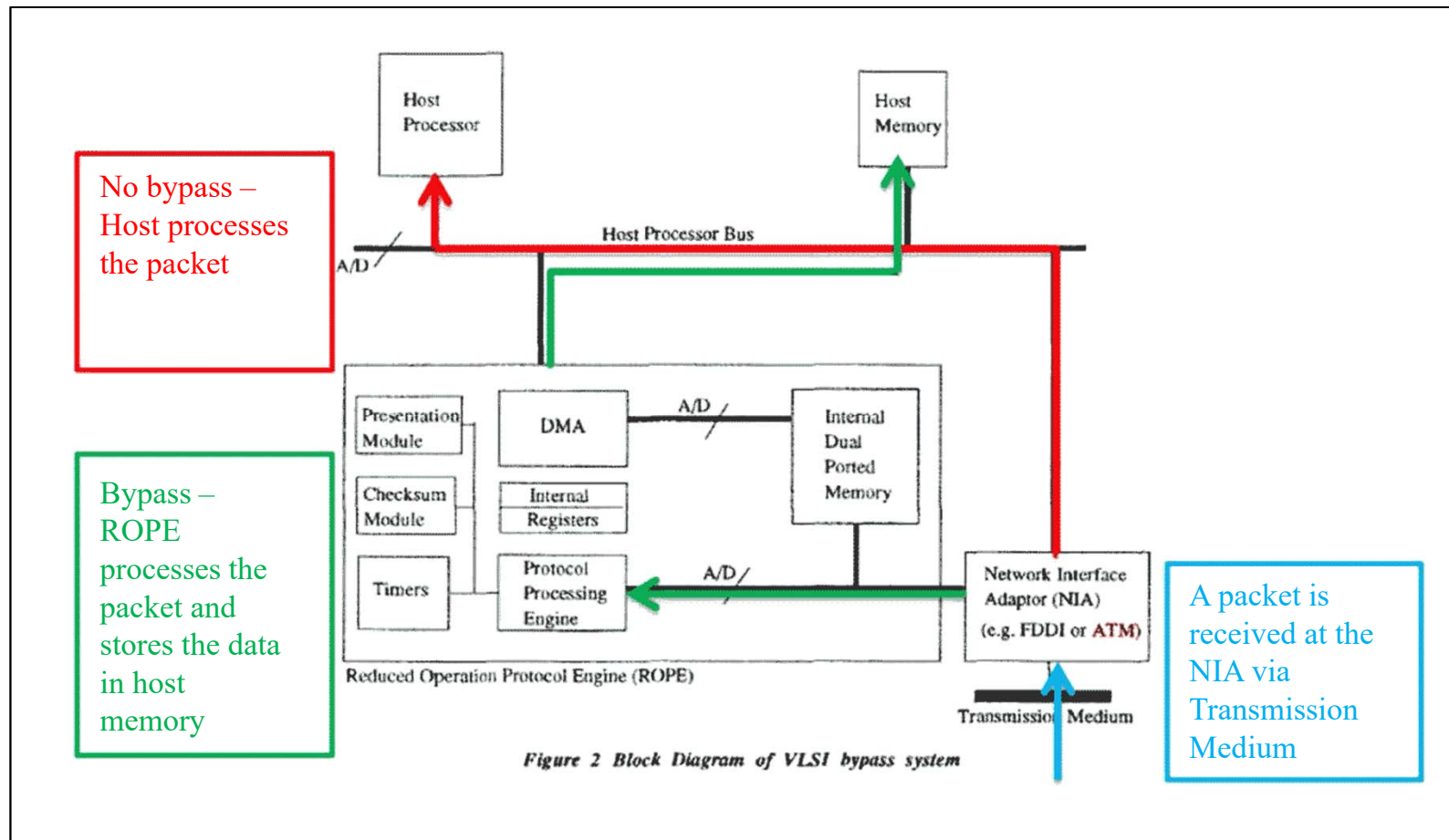


◆ In the ATM world, fragmentation is called segmentation



Ex. 1006.426 (Tanenbaum96);
See also Ex. 1223.020 (1410 Lin Reply Decl.).

This teaches lower-layer segmentation/reassembly on the NIA



Ex. 1223.019-.020 (1410 Lin Reply) at ¶ 28 (excerpting and annotating Figure 2 from Ex. 1015.007 (Thia) with red, green, blue annotations and red shading); see also Paper 42 (1410 Reply) at 17-18.

Explanation of network layer (IP) fragmentation

THIRD EDITION COMPUTER NETWORKS

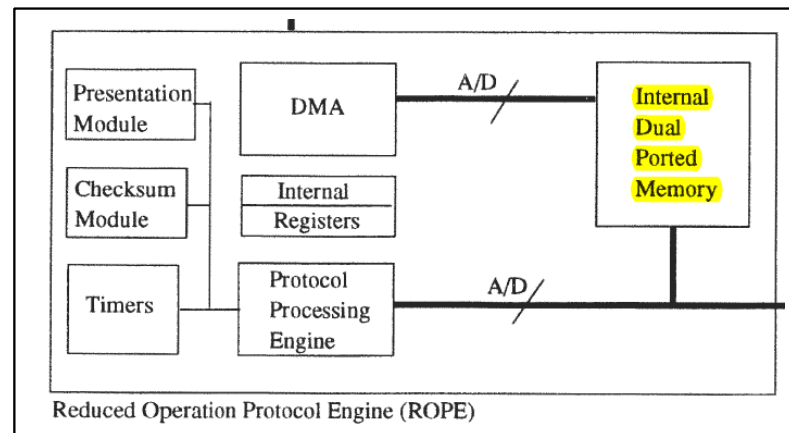
Communication in the Internet works as follows. The transport layer takes data streams and breaks them up into datagrams. In theory, datagrams can be up to 64 Kbytes each, but in practice they are usually around 1500 bytes. Each datagram is transmitted through the Internet, possibly being fragmented into smaller units as it goes. When all the pieces finally get to the destination machine, they are reassembled by the network layer into the original datagram. This datagram is then handed to the transport layer, which inserts it into the receiving process' input stream.



Ex. 1006.431 (Tanenbaum96); See also Paper 42 (1410 Reply) at 17-18; Ex.1223.017-.018 (1410 Lin Reply Decl.).

Disclosure for *transmitting* a packet fails to rebut disclosure of re-assembly

- 2) For subsequent bypassable packets, the host processor initiates the BYPASS_DMA procedure which checks for free buffer space in the bypass chip and programs the DMA by sending the starting address pointer where the PDU is located, and its total length. The destination address is supplied by the bypass chip. Arbitration for the host processor bus between the host and DMA is provided by the DMAreq and DMAack lines. DMA transfers the PDU into the internal dual-ported SRAM (Static RAM). Buffers are pre-allocated in fixed sizes and are accessed by a simple round robin scheme using a set of buffer pointers.



Ex. 1015.009 (Thia).

Ex. 1015.007 (Thia) (Fig. 2).

880 Patent: Disputes

3. The prior art combinations disclose the limitations of the challenged claims of the 880 Patent
 - a. The prior art combination renders obvious “an operation code” (claims 1, 17, 32, 34, 45)
 - b. This discloses a “re-assembler” on, or “re-assembly” by, a network interface (claims 32, 41, 43)
 - c. **A “flow key” that includes a “first hop medium access control (MAC) layer address” would have been obvious (claim 32)**
 - d. The prior art combination discloses storing the “header portion in a header buffer” if the “header conforms to the TCP protocol” (claim 32)
 - e. The prior art combination discloses a “processor” for TCP processing (claims 1, 32, 41, 43)

880 Patent: Claim 32

(12) **United States Patent**
Boucher et al.

(10) Patent No.: **US 8,131,880 B2**
(45) Date of Patent: ***Mar. 6, 2012**

(54) **INTELLIGENT NETWORK INTERFACE DEVICE AND SYSTEM FOR ACCELERATED COMMUNICATION**

(75) Inventors: **Laurence B. Boucher**, Saratoga, CA (US); **Stephen E. J. Blightman**, San Jose, CA (US); **Peter K. Craft**, San Francisco, CA (US); **David A. Higgen**, Saratoga, CA (US); **Clive M. Philbrick**, San Jose, CA (US); **Daryl D. Starr**, Milpitas, CA (US)

(73) Assignee: **Alacritech, Inc.**, San Jose, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 2126 days.
This patent is subject to a terminal disclaimer.

(21) Appl. No.: **10/601,237**

(22) Filed: **Jun. 19, 2003**

(65) **Prior Publication Data**
US 2004/0062246 A1 Apr. 1, 2004

Related U.S. Application Data

(63) Continuation of application No. 10/005,536, filed on Nov. 7, 2001, now Pat. No. 7,167,926, and a continuation-in-part of application No. 09/514,425, filed on Feb. 28, 2000, now Pat. No. 6,427,171, and a continuation-in-part of application No. 09/464,283, filed on Dec. 15, 1999, now Pat. No. 6,427,173, and a continuation of application No. 09/384,792, filed on Aug. 27, 1999, now Pat. No. 6,434,620, and a continuation-in-part of application No. 09/067,544, filed on Apr. 27, 1998, now Pat. No. 6,226,680, and a continuation-in-part of application No. 09/141,713, filed on Aug. 28, 1998, now Pat. No. 6,389,479.

(60) Provisional application No. 60/098,296, filed on Aug. 27, 1998, provisional application No. 60/061,809, filed on Oct. 14, 1997.

(51) Int. Cl. **G06F 15/16** (2006.01)

(52) U.S. CL. **709/250**

(58) **Field of Classification Search** **709/238, 709/250**
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS
4,366,538 A 12/1982 Johnson et al. 364/200 (Continued)

FOREIGN PATENT DOCUMENTS
WO WO/98/19412 5/1998 (Continued)

OTHER PUBLICATIONS
U.S. Appl. No. 60/053,240, by Jolitz et al. (listed filing date Jul. 18, 1997). (Continued)

ABSTRACT
An intelligent network interface card (NIC) or communication processing device (CPD) works with a host computer for data communication. The device provides a fast-path that avoids protocol processing for most messages, greatly accelerating data transfer and offloading time-intensive processing tasks from the host CPU. The host retains a fallback processing capability for messages that do not fit fast-path criteria, with the device providing assistance such as validation even for slow-path messages, and messages being selected for either fast-path or slow-path processing. A context for a connection is defined that allows the device to move data, free of headers, directly to or from a destination or source in the host. The context can be passed back to the host for message processing by the host. The device contains specialized hardware circuits that are much faster at their specific tasks than a general purpose CPU. A preferred embodiment includes a trio of pipelined processors devoted to transmit, receive and utility processing, providing full duplex communication for four Fast Ethernet nodes.

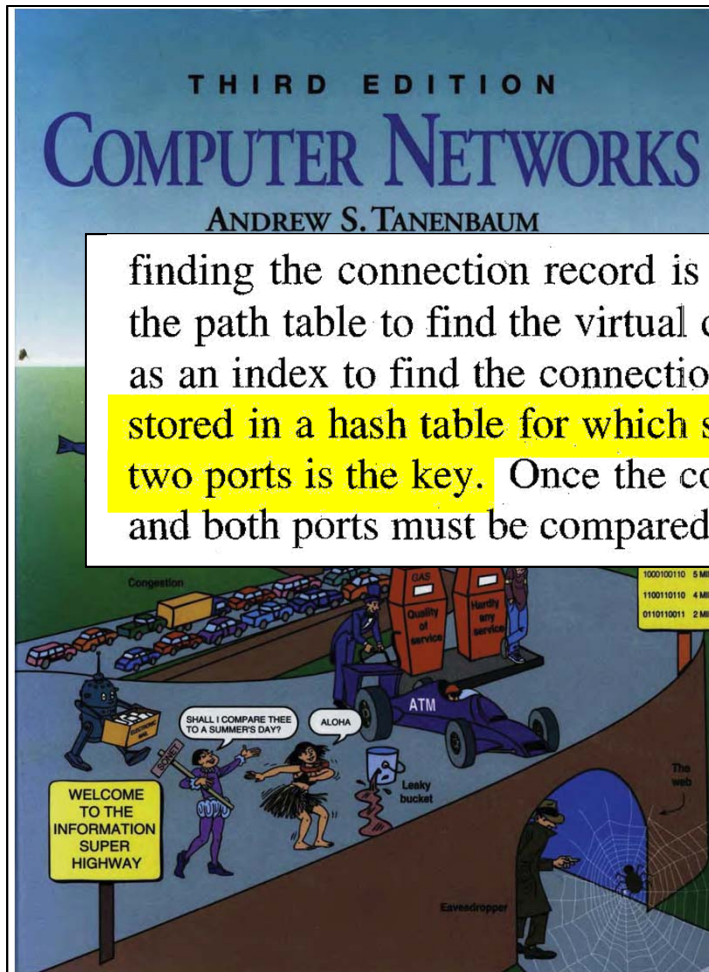
60 Claims, 40 Drawing Sheets

32. A method of transferring a packet received at a network interface from a network to a host computer system, comprising:

- receiving a packet from a network at a network interface of a host computer system;
- parsing a header portion of said packet to extract an identifier of a source entity and an identifier of a destination entity;
- generating a flow key from said source identifier and said destination identifier to identify a communication flow comprising said packet, **wherein said flow key includes a TCP connection for the communication flow and a first hop medium access control (MAC) layer address;**
- determining whether a header in said header portion conforms to a pre-selected protocol;
- storing said flow key in a database;
- associating an operation code with said packet, wherein said operation code identifies a status of said packet;
- storing said packet in a packet memory;
- if said header conforms to the TCP protocol:
- storing a data portion of said packet in a re-assembly buffer;
- storing said header portion in a header buffer; and
- processing, by the network interface, said packet according to the TCP connection.

Ex. 1001 (880 Patent), Claim 32.

Tanenbaum96 discloses flow key comprising the TCP/IP socket pair



finding the connection record is easy: the *VPI* field can be used as an index into the path table to find the virtual circuit table for that path and the *VCI* can be used as an index to find the connection record. 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.

Ex. 1006.585 (Tanenbaum96);
See Paper 1 (1410 Petition) at 47-48;
Ex. 1003.093-.095 (1410 Lin Decl.);
See also Petition 1 (1410 Petition) at 30, 48.

It would be obvious to include header information relevant to the connection

UNITED STATES PATENT AND TRADEMARK OFFICE	
BEFORE THE PATENT TRIAL AND APPEAL BOARD	
INTEL CORPORATION Petitioner	Further, it would have been obvious to a person of ordinary skill in the art to have a flow key additionally include a next-hop MAC layer address. A person of ordinary skill in the art would know that a flow key can comprise any combination of characters or numbers, and that it would make sense to have this combination comprise information relevant to the connection in order to verify the flow key against the information relevant to the connection during lookup. See Ex.1006,
ALACRITECH, INC. Patent Owner	
Case IPR. No. Unassigned U.S. Patent No. 8,131,880	
Title: INTELLIGENT NETWORK INTERFACE DEVICE AND SYSTEM FOR ACCELERATED COMMUNICATION	
DECLARATION OF BILL LIN IN SUPPORT OF PETITION FOR INTER PARTES REVIEW OF U.S. PATENT NO. 8,131,880 UNDER 37 C.F.R. § 1.68	
Mail Stop "PATENT BOARD" Patent Trial and Appeal Board U.S. Patent and Trademark Office P.O. Box 1450 Alexandria, VA 22313-1450	the correct record has been found.”). As explained in paragraphs 30 and 31 of my declaration, a next-hop MAC layer address indicates the physical location a packet is first routed on its way to its final destination for the connection and is therefore information relevant to the connection. ¹
INTEL Ex.1003.001	

Ex. 1003.095 (1410 Lin Decl.);
See also Paper 1 (1410 Petition) at 49.

MAC layer address is relevant to the connection

Ex. 1003 ¶¶30-31. Moreover, a POSA would understand that it would be useful to store the first hop MAC layer address in the connection record in order to send packets back out on the network, for example in the situation where acknowledgements are sent. Before the acknowledgement can reach its destination, however, it must first go through the gateway that corresponds to the first-hop MAC layer address. Thus, the first hop MAC layer address is a candidate for including in a flow key that a POSA would have found obvious.

Ex. 1223.014-.015 (1410 Lin Decl.) at ¶ 18;

See also Paper 1 (1410 Petition) at 49;

Ex. 1003.095 (1410 Lin Decl.); Paper 42 (1410 Reply) at 10-12.

The pool of fields to include in a flow key is finite and small

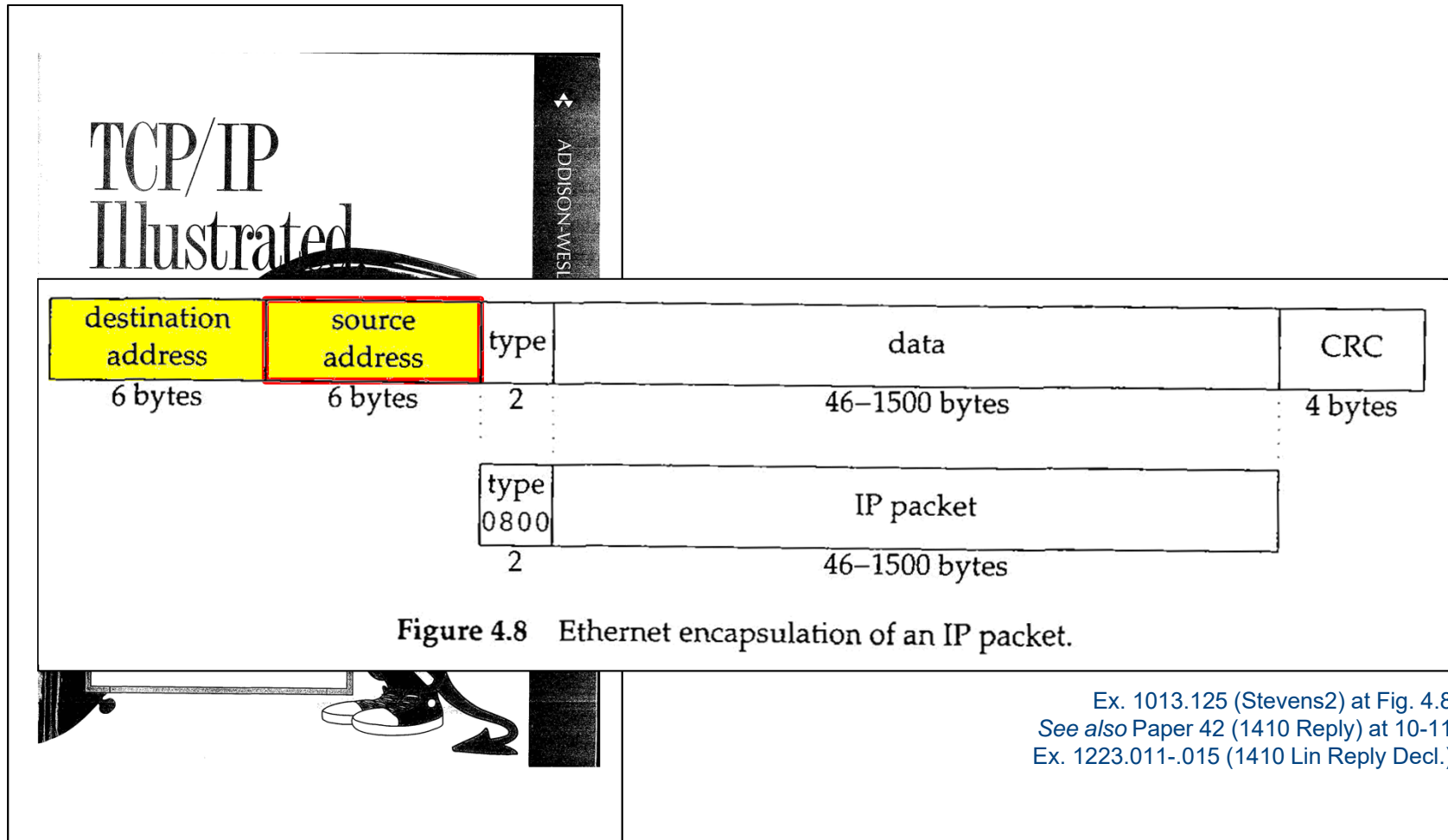
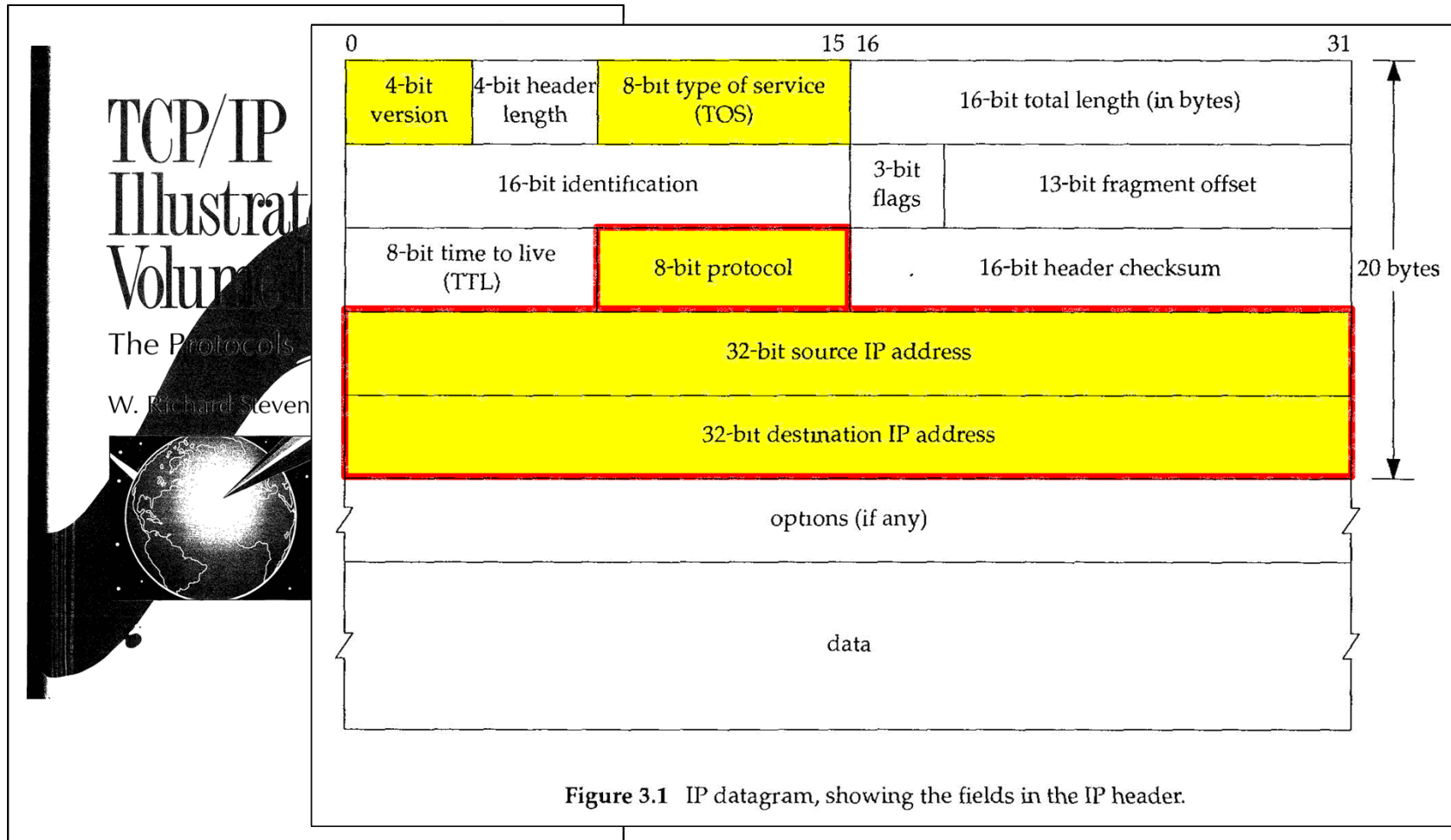


Figure 4.8 Ethernet encapsulation of an IP packet.

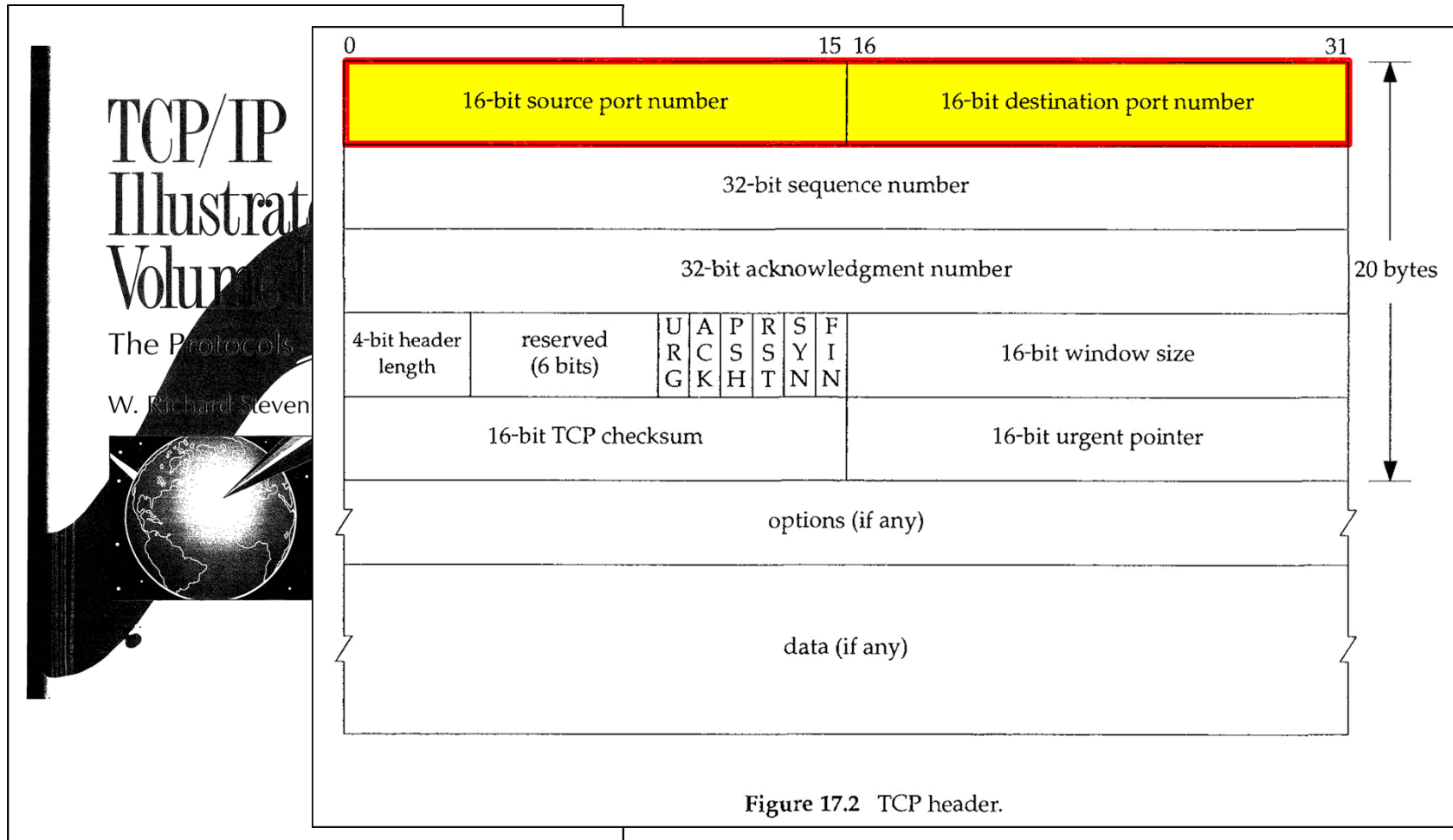
Ex. 1013.125 (Stevens2) at Fig. 4.8;
See also Paper 42 (1410 Reply) at 10-11;
Ex. 1223.011-.015 (1410 Lin Reply Decl.).

The pool of fields to include in a flow key is finite and small



Ex. 1008.058 (Stevens1) at Fig. 3.1;
See also Paper 42 (1410 Reply) at 10-11; Ex. 1223.011-.015 (1410 Lin Reply Decl.).

The pool of fields to include in a flow key is finite and small



Ex. 1008.249 (Stevens1) at Fig. 17.2;
 See also Paper 42 (1410 Reply) at 10-11; Ex. 1223.011-015 (1410 Lin Reply Decl.).

Dr. Almeroth opined that a MAC layer address is not required to infringe

RESTRICTED – ATTORNEYS' EYES ONLY – INTEL ADDENDUM/RESTRICTED CONFIDENTIAL – SOURCE CODE

7.11.2 Flow Identification and RSC Context Matching

TCP/IP packet's flow is identified by its four Source / Destination TCP port numbers. The Identification fields stored in the active RSC is done in two phases:

- Hash Compare — Hardware computes the hash value is stored in the RSC context. The hash value of the compare logic. The hash value of the hash values of all RSC contexts. No match there is no valid context of the same frame.
- Perfect Match — Hardware checks the first step with the received frame.
 - A match between the two means that the RSC context is valid.
 - Mismatch between the two indicates a collision of the collided RSC.
- In any case of context mismatch, a new RSC context is created as described in Section 7.11.3.
- If the packet's flow matches an active RSC context, the RSC context is appended to the existing RSC as described in Section 7.11.3.

(Niantic Datasheet at § 7.11.2; see also Sageville Datasheet at § 7.11.2; Broadwell DE Datasheet at § 7.9.2; Denver Datasheet at § 7.9.2; through 200:19; *id.* 217:15-25; Sarangam Dep. Tr. 23-24)

61. Claim 32(c): “generating a flow key from said source identifier and said destination identifier to identify a communication flow comprising

a TCP connection for the communication flow and a first hop medium access control (MAC) layer address” After extracting the source and destination TCP ports and the source and destination IP addresses, the accused RSC products generate a hash value based on those four tuples. That hash value is then used to identify an active RSC context, and is thus a flow key or “context identifier” as construed by the Court:

50

61. Claim 32(c): “generating a flow key from said source identifier and said destination identifier to identify a communication flow comprising said packet, wherein said flow key includes a TCP connection for the communication flow and a first hop medium access control (MAC) layer address” After extracting the source and destination TCP ports and the source and destination IP addresses, the accused RSC products generate a hash value based on those four tuples. That hash value is then used to identify an active RSC context, and is thus a flow key or “context identifier” as construed by the Court:

Ex. 1249.005 (Almeroth Infrgmt. Rpt.) at ¶ 61; See also Paper 42 (1410 Reply) at 13.

880 Patent: Disputes

3. The prior art combinations disclose the limitations of the challenged claims of the 880 Patent
 - a. The prior art combination renders obvious “an operation code” (claims 1, 17, 32, 34, 45)
 - b. This discloses a “re-assembler” on, or “re-assembly” by, a network interface (claims 32, 41, 43)
 - c. A “flow key” that includes a “first hop medium access control (MAC) layer address” would have been obvious (claim 32)
 - d. **The prior art combination discloses storing the “header portion in a header buffer” if the “header conforms to the TCP protocol” (claim 32)**
 - e. The prior art combination discloses a “processor” for TCP processing (claims 1, 32, 41, 43)

880 Patent: Claim 32

(12) **United States Patent**
Boucher et al.

(10) Patent No.: **US 8,131,880 B2**
(45) Date of Patent: ***Mar. 6, 2012**

(54) **INTELLIGENT NETWORK INTERFACE DEVICE AND SYSTEM FOR ACCELERATED COMMUNICATION**

(75) Inventors: **Laurence B. Boucher**, Saratoga, CA (US); **Stephen E. J. Blightman**, San Jose, CA (US); **Peter K. Craft**, San Francisco, CA (US); **David A. Higgen**, Saratoga, CA (US); **Clive M. Philbrick**, San Jose, CA (US); **Daryl D. Starr**, Milpitas, CA (US)

(73) Assignee: **Alacritech, Inc.**, San Jose, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 2126 days.
This patent is subject to a terminal disclaimer.

(21) Appl. No.: **10/601,237**

(22) Filed: **Jun. 19, 2003**

(65) **Prior Publication Data**
US 2004/0062246 A1 Apr. 1, 2004

Related U.S. Application Data

(63) Continuation of application No. 10/005,536, filed on Nov. 7, 2001, now Pat. No. 7,167,926, and a continuation-in-part of application No. 09/514,425, filed on Feb. 28, 2000, now Pat. No. 6,427,171, and a continuation-in-part of application No. 09/464,283, filed on Dec. 15, 1999, now Pat. No. 6,427,173, and a continuation of application No. 09/384,792, filed on Aug. 27, 1999, now Pat. No. 6,434,620, and a continuation-in-part of application No. 09/067,544, filed on Apr. 27, 1998, now Pat. No. 6,226,680, and a continuation-in-part of application No. 09/141,713, filed on Aug. 28, 1998, now Pat. No. 6,389,479.

(60) Provisional application No. 60/098,296, filed on Aug. 27, 1998, provisional application No. 60/061,809, filed on Oct. 14, 1997.

(51) Int. Cl. **G06F 15/16** (2006.01)

(52) U.S. CL. **709/250**

(58) **Field of Classification Search** **709/238, 709/250**
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS
4,366,538 A 12/1982 Johnson et al. 364/200 (Continued)

FOREIGN PATENT DOCUMENTS
WO WO/98/19412 5/1998 (Continued)

OTHER PUBLICATIONS
U.S. Appl. No. 60/053,240, by Jolitz et al. (listed filing date Jul. 18, 1997). (Continued)

Primary Examiner — Jerry Dennison
(74) *Attorney, Agent, or Firm* — Silicon Edge Law Group LLP, Mark A. Lauer

(57) **ABSTRACT**
An intelligent network interface card (NIC) or communication processing device (CPD) works with a host computer for data communication. The device provides a fast-path that avoids protocol processing for most messages, greatly accelerating data transfer and offloading time-intensive processing tasks from the host CPU. The host retains a fallback processing capability for messages that do not fit fast-path criteria, with the device providing assistance such as validation even for slow-path messages, and messages being selected for either fast-path or slow-path processing. A context for a connection is defined that allows the device to move data, free of headers, directly to or from a destination or source in the host. The context can be passed back to the host for message processing by the host. The device contains specialized hardware circuits that are much faster at their specific tasks than a general purpose CPU. A preferred embodiment includes a trio of pipelined processors devoted to transmit, receive and utility processing, providing full duplex communication for four Fast Ethernet nodes.

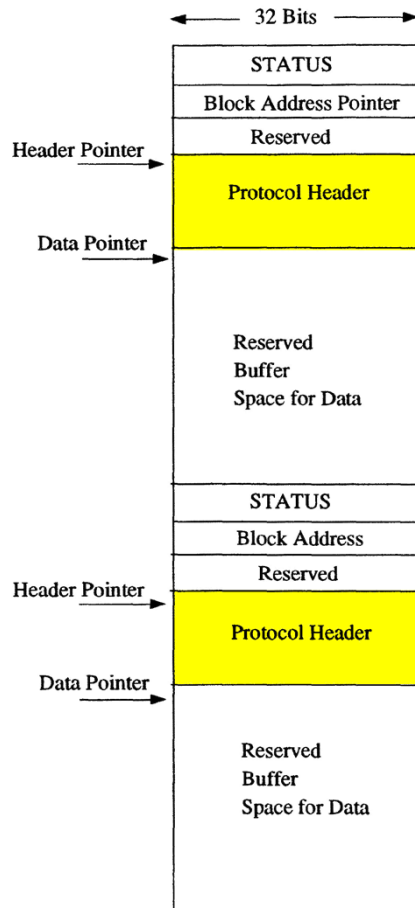
60 Claims, 40 Drawing Sheets

32. A method of transferring a packet received at a network interface from a network to a host computer system, comprising:

- receiving a packet from a network at a network interface of a host computer system;
- parsing a header portion of said packet to extract an identifier of a source entity and an identifier of a destination entity;
- generating a flow key from said source identifier and said destination identifier to identify a communication flow comprising said packet, wherein said flow key includes a TCP connection for the communication flow and a first hop medium access control (MAC) layer address;
- determining whether a header in said header portion conforms to a pre-selected protocol;
- storing said flow key in a database;
- associating an operation code with said packet, wherein said operation code identifies a status of said packet;
- storing said packet in a packet memory;
- if said header conforms to the TCP protocol:
- storing a data portion of said packet in a re-assembly buffer;
- storing said header portion in a header buffer; and
- processing, by the network interface, said packet according to the TCP connection.

Ex. 1001 (880 Patent), Claim 32.

880 Patent: Storing said header portion in a header buffer



Ex. 1015.011 (Thia) at Fig. 4.

A buffer is a memory portion for the storage of data while it is being processed or transferred. Accordingly, a person of ordinary skill in the art would understand that a buffer is a portion of memory. A portion of memory that stores data would be a data buffer, a portion of memory that stores headers would be a header buffer, and so on.

Ex. 1003.111 (1410 Lin Decl.); See also Paper 1 (1410 Petition) at 58-60.

Claims do not recite a “separate” header buffer

Case No. IPR2017-1410
U.S. Patent No. 8,131,880

C. The Combination Does Not Show or Suggest “if said header conforms to the TCP protocol . . . storing said header portion in a header buffer”

The Petition fails to suggest the limitation “if header portion in a header buffer in a figure from Thia and portion of memory.” (The “Reserved Buffer Space for memory, which stores PDUs “DMA”—they are not the for that matter). (Ex. 20 packets, the host processor free buffer space in the by

portion of memory.” (Petition, 58-60.) However, the “Protocol Header” and “Reserved Buffer Space for Data” in Fig. 4 are, if anything, the ROPE chip’s packet memory, which stores PDUs for transfer to the host via direct memory access or “DMA”—they are not the claimed separate “header buffer” (or “re-assembly buffer” for that matter). (Ex. 2026, ¶ 147; Ex. 1015.009 (“For subsequent bypassable

address pointer where the PDU is located, and its total length.”); Petition, 55-57 (identifying the same memory locations as comprising the claimed “packet memory”).)

Tellingly, Petitioners allege the claimed “storing said packet in a packet memory” step is met by copying the header and data of an incoming PDU to the “Protocol Header” and “Reserved Buffer Space for Data” blocks in memory shown in Fig. 4 of Thia. (*Id.*) The “storing said header portion in a header buffer,” however,

storing said header portion in a header buffer; and

880 Patent, Claim 32;
See also Paper 42 (1410 Reply) at 16-17.

880 Patent: Disputes

3. The prior art combinations disclose the limitations of the challenged claims of the 880 Patent
 - a. The prior art combination renders obvious “an operation code” (claims 1, 17, 32, 34, 45)
 - b. This discloses a “re-assembler” on, or “re-assembly” by, a network interface (claims 32, 41, 43)
 - c. A “flow key” that includes a “first hop medium access control (MAC) layer address” would have been obvious (claim 32)
 - d. The prior art combination discloses storing the “header portion in a header buffer” if the “header conforms to the TCP protocol” (claim 32)
 - e. **The prior art combination discloses a “processor” for TCP processing (claims 1, 32, 41, 43)**

880 Patent: Claim 1

(12) **United States Patent**
Boucher et al. (10) Patent No.: **US 8,131,880 B2**
 (45) Date of Patent: ***Mar. 6, 2012**

(54) **INTELLIGENT NETWORK INTERFACE DEVICE AND SYSTEM FOR ACCELERATED COMMUNICATION**

(75) Inventors: **Laurence B. Boucher**, Saratoga, CA (US); **Stephen E. J. Blightman**, San Jose, CA (US); **Peter K. Craft**, San Francisco, CA (US); **David A. Higgen**, Saratoga, CA (US); **Clive M. Philbrick**, San Jose, CA (US); **Daryl D. Starr**, Milpitas, CA (US)

(73) Assignee: **Alacritech, Inc.**, San Jose, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 2126 days. This patent is subject to a terminal disclaimer.

(21) Appl. No.: **10/601,237**

(22) Filed: **Jun. 19, 2003**

(65) **Prior Publication Data**
 US 2004/0062246 A1 Apr. 1, 2004

Related U.S. Application Data

(63) Continuation of application No. 10/905,536, filed on Nov. 7, 2001, now Pat. No. 7,167,926, and a continuation-in-part of application No. 09/514,425, filed on Feb. 28, 2000, now Pat. No. 6,427,171, and a continuation-in-part of application No. 09/464,283, filed on Dec. 15, 1999, now Pat. No. 6,427,173, and a continuation of application No. 09/384,792, filed on Aug. 27, 1999, now Pat. No. 6,434,620, and a continuation-in-part of application No. 09/067,544, filed on Apr. 27, 1998, now Pat. No. 6,226,680, and a continuation-in-part of application No. 09/141,713, filed on Aug. 28, 1998, now Pat. No. 6,389,479.

(60) Provisional application No. 60/098,296, filed on Aug. 27, 1998, provisional application No. 60/061,809, filed on Oct. 14, 1997.

(51) Int. Cl. **G06F 15/16**

(52) U.S. Cl.

(58) Field of Classification

See application file for

(56) **References**

U.S. PATENT
 4,366,538 A 12/1982 (Cont)

FOREIGN PATENT
 WO WO/98/19412 (Cont)

OTHER PUBLICATIONS
 U.S. Appl. No. 60/053,240, by John D. Boucher, et al., filed on Oct. 14, 1997. (Cont)

Primary Examiner — Jerry D. ...
(74) Attorney, Agent, or Firm — L.L.P.; Mark A. Lauer

(57) **ABSTRACT**
 An intelligent network interface device (INID) for processing data communication. The device avoids protocol processing for generating data transfer and off-host tasks from the host CPU. The device provides a fast-path for slow-path messages, and either fast-path or slow-path processing is defined that allows a context to be passed to the host. The device includes hardware circuits that are much faster than a general purpose CPU. A preferred embodiment includes pipelined processors devoted to processing, providing full Fast Ethernet nodes.

60 Claims, 40

1. A method of transferring a packet to a host computer system, wherein the packet is received at a communication device from a network, comprising:

- parsing a header portion of a first packet received at a network interface for the host computer system to determine if said first packet conforms to a TCP protocol;
- generating a flow key to identify a first communication flow that includes said first packet, wherein said flow key includes a TCP connection for the communication flow;
- associating an operation code with said first packet, wherein said operation code indicates a status of said first packet, including whether said packet is a candidate for transfer to the host computer system that avoids processing said header portion by the host computer system in accordance with said TCP protocol; and
- processing, by the network interface, said packet according to the TCP connection, including updating a control block representing the TCP connection on the network interface.

Ex. 1001 (880 Patent), Claim 1.

880 Patent: Claims 41 and 43

(12) **United States Patent**
Boucher et al.

(10) Patent No.: US 8,131,880 B2
(45) Date of Patent: *Mar. 6, 2012

41. An apparatus for transferring a packet to a host computer system, comprising:

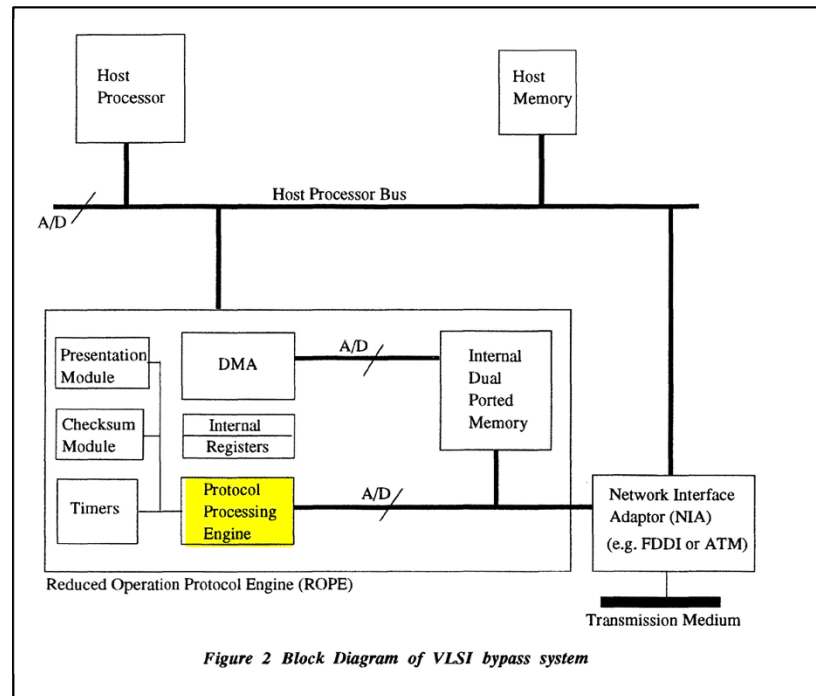
- a traffic classifier, disposed in a network interface for the host computer system, configured to classify a first packet received from a network by a communication flow that includes said first packet;
- a packet memory, disposed in the network interface, configured to store said first packet;
- a packet batching module, disposed in the network interface, configured to determine whether another packet in said packet memory belongs to said communication flow;
- a flow re-assembler, disposed in the network interface, configured to re-assemble a data portion of said first packet with a data portion of a second packet in said communication flow; and
- a processor, disposed in the network interface, that maintains a TCP connection for the communication flow, the TCP connection stored as a control block on the network interface.

43. A computer system for receiving a packet from a network, comprising:

- a memory configured to store packets received from a network; and
- a network interface for the computer system, the network interface configured to receive a first packet from said network, the network interface comprising:
 - a parser configured to extract information from a header portion of a first packet;
 - a flow manager configured to examine said information;
 - a flow database configured to store an identifier of a first communication flow comprising multiple packets, including said first packet; and
 - a re-assembler for storing data portions of said multiple packets without header portions in a first portion of said memory; and
- a processor for processing said first packet and for maintaining a TCP connection for the communication flow, the TCP connection stored as a control block on the network interface.

Ex. 1001 (880 Patent), Claims 41, 43.

The prior art combination renders obvious TCP processing



Ex.1015 (Thia) at Fig. 2; See also Paper 1; (1409 Petition) 51-57; Paper 1 (1410 Petition) at 60-65.

towards TCP/IP, a POSA would have been motivated to combine Thia with the TCP/IP protocol teachings of Tanenbaum96. See Ex.1003 ¶114. As described by

Paper 1 (1410 Petition) at 34. See also Paper 1 (1409 Petition) 32.

880 Patent: Disputes

1. A POSA would have combined Thia and Tanenbaum96 (and Nahum)
2. Thia and Nahum are enabling
3. The prior art combinations disclose the limitations of the challenged claims of the 880 Patent
4. **Motions to Amend 880 Patent should be denied**

880 Patent: Motions to Amend

- 2017IPR-01409, IPR2017-01736, IPR2018-00338: Amending all challenged claims except for claim 8, which is cancelled
- 2017IPR-01410, IPR2017-01737, IPR2018-00339: Amending all challenged claims

880 Patent: Disputes

4. Motions to Amend 880 Patents should be denied

- a) **PO has not met its burden of production under 35 U.S.C. § 316(d) due to its failure to provide adequate written description support**
- b) The substitute claims are indefinite
- c) The prior art combinations disclose the limitations of the substitute claims

PO only provides string citations

Case No. IPR2017-01410
U.S. Patent No. 8,131,880

if said header conforms to the TCP protocol: storing a data portion of said packet in a re-assembly buffer;	<i>See, e.g.</i> , Ex. 2025 at Abstract, ¶¶ [0115], [0271], Fig. 2, Cl. 33.
storing said header portion in a header buffer, <u>wherein the header buffer is separate from said packet memory;</u>	<i>See, e.g.</i> , Ex. 2025 at Abstract, ¶¶ [0115], [0271], Fig. 2, Cl. 33.
<u>re-assembling the data portion of said packet with a data portion of another packet in the communication flow;</u> and	<i>See, e.g.</i> , Ex. 2025 at Abstract, ¶¶ [0115]-[0116], [0271], Fig. 2, Cls. 42, 44, 59.
processing, by the network interface, said packet <u>and said other packet</u> according to the TCP connection.	<i>See, e.g.</i> , Ex. 2025 at Abstract, ¶¶ [0013], [0074], [0080], [0082]-[0083], [0115], Figs. 1, 2, Cl. 44.
Proposed Claim 80	
80. (proposed substitute for claim 34) The method of claim <u>[321] 79</u> , further comprising storing said operation code in a control memory.	<i>See, e.g.</i> , support cited for proposed claim 79; <i>see also</i> Ex. 2025 at Cl. 35.
Proposed Claim 81	
81. (proposed substitute for claim 35) The method of claim <u>[321] 79</u> , further comprising storing a flow number of said packet in a flow memory, wherein said flow number comprises an index of said flow key in said database.	<i>See, e.g.</i> , support cited for proposed claim 79; <i>see also</i> Ex. 2025 at Cl. 36.
Proposed Claim 82	
82. (proposed substitute for claim 37) The method of claim <u>[321] 79</u> , wherein the host computer system comprises multiple processors for processing network packets in accordance with the TCP protocol.	<i>See, e.g.</i> , support cited for proposed claim 79; <i>see also</i> Ex. 2025 at Cl. 38.
Proposed Claim 83	
83. (proposed substitute for claim 38) The method of claim <u>[371] 82</u> , further comprising: receiving a second packet at said network interface, wherein said second packet is part of a second communication flow; and identifying a processor in the host computer system to process said second packet.	<i>See, e.g.</i> , support cited for proposed claim 79; <i>see also</i> Ex. 2025 at Cl. 39.
	<i>See, e.g.</i> , support cited for proposed claim 79; <i>see also</i>

ii

Paper 20 (1410 Motion to Amend) at ii.

Case No. IPR2017-01410
U.S. Patent No. 8,131,880

status of said packet;	assistance”); § 5.1 (“Design Overview”).
storing said packet in a packet memory;	<i>See, e.g.</i> , Ex. 2019 at § 5.1 (“Design Overview”); p. 72 (“GENERAL DESCRIPTION”); p. 117-18 (“FRAME RECEIVE SEQUENCER (RevX)”).
if said header conforms to the TCP protocol: storing a data portion of said packet in a re-assembly buffer;	<i>See, e.g.</i> , Ex. 2019 at § 3.1 (“Receive Interface”); § 4.6.2.1 (“Receive overview”).
storing said header portion in a header buffer, <u>wherein the header buffer is separate from said packet memory;</u>	<i>See, e.g.</i> , Ex. 2019 at § 3.1 (“Receive Interface”); § 4.6.2.1 (“Receive overview”); § 5.1 (“Design Overview”); p. 117-18 (“FRAME RECEIVE SEQUENCER (RevX)”).
<u>re-assembling the data portion of said packet with a data portion of another packet in the communication flow;</u> and	<i>See, e.g.</i> , Ex. 2019 at § 3.1 (“Receive Interface”); § 5.1 (“Design Overview”).
processing, by the network interface, said packet <u>and said other packet</u> according to the TCP connection.	<i>See, e.g.</i> , Ex. 2019 at § 2 (“Summary of the Invention”); § 2.1.1 (“Only Support TCP/IP”); § 2.1.3 (“Two modes of operation”); § 3.1 (“Receive Interface”).
Proposed Claim 80	
80. (proposed substitute for claim 34) The method of claim <u>[321] 79</u> , further comprising storing said operation code in a control memory.	<i>See, e.g.</i> , Ex. 2019 at § 2.1.5 (“TCP hardware assistance”); § 5.1 (“Design Overview”).
Proposed Claim 81	
81. (proposed substitute for claim 35) The method of claim <u>[321] 79</u> , further comprising storing a flow number of said packet in a flow memory, wherein said flow number comprises an index of said flow key in said database.	<i>See, e.g.</i> , Ex. 2019 at § 2.1.1 (“Only Support TCP/IP”); § 2.1.4 (“The TCP cache”); § 5.1 (“Design Overview”).

vii

Paper 20 (1410 Motion to Amend) at vii.

See Paper 38 (1409 Opp. to Motion to Amend) at 2-9; Paper 38 (1410 Opp. to Motion to Amend) at 2-8.

PO's citations do not identify a “packet memory”; just general purpose RAM

[0838] FIG. 50 is a diagram of a FRAME RECEIVE SEQUENCER (RcvX). The receive sequencer (RcvSeq) analyzes and manages incoming packets, stores the result in **DRAM buffers**, then notifies the processor through the receive queue (RcvQ) mechanism. The process begins when a buffer descriptor is available at the output of the FreeQ. RcvSeq issues a request to the Qmg which responds by supplying the buffer descriptor to RcvSeq. RcvSeq then waits for a receive packet. The Mac, network, transport and session information is analyzed as each byte is received and stored in the assembly register (AssyReg). When four bytes of information is available, RcvSeq requests a write of the data to the **SRAM**. When sufficient data has been stored in the **SRAM based receive FIFO**, a DRAM write request is issued to Xwr. The process continues until the entire packet has been received at which point RcvSeq stores the results of the packet analysis in the beginning of the DRAM buffer. Once the buffer and status have both been stored, RcvSeq issues a write-queue request to Qmg. Qmg responds by storing a buffer descriptor and a status vector provided by RcvSeq. The process then repeats. If RcvSeq detects the arrival of a packet before a free buffer is available, it ignores the packet and sets the FrameLost status bit for the next received packet.

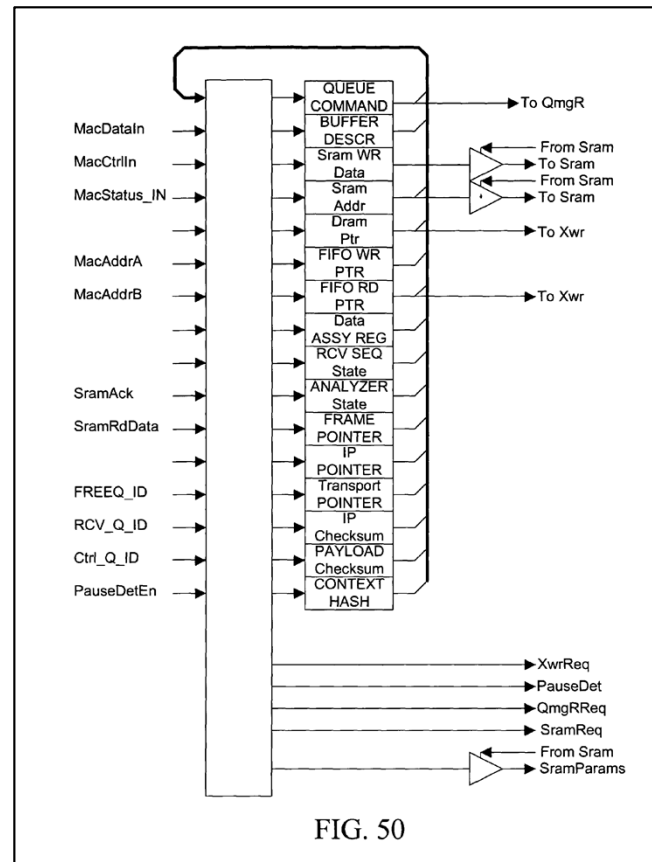


FIG. 50

Ex. 2025.092 (880 App. Pub.) at ¶ [0838];
See Paper 50 (1409 Sur-Reply to Motion to Amend) at 2-3;
Paper 50 (1410 Sur-Reply to Motion to Amend) at 2-3.

Ex. 2025.037 (880 App. Pub.) at Fig. 50.

PO's citations do not identify a header buffer separate from packet memory

storing said header portion in a header buffer, wherein the header buffer is separate from said packet memory;

See, e.g., Ex. 2025 at Abstract, ¶¶ [0115], [0271], Fig. 2, Cl. 33.

Paper 20 (1410 Motion to Amend), App'x A at ii.

[0115] As shown in FIG. 2, the fast-path flow puts a header such as HEADER A 90 into a header buffer that is then forwarded to the host. HEADER A contains status 92 that has been generated by the INIC and TCP/SMB headers 94 that can be used by the host to determine what further data is following and allocate the necessary host buffers, which are then passed back to the INIC as data buffer descriptors 96 via a command to the INIC. The INIC then fills these buffers from data it was accumulating on the card and notifies the host by sending a response to the command. Alternatively, the fast-path may receive a header and data that is a complete request, but that is also too large for a header buffer. This results in a header and data buffer being passed to the host. This latter flow is similar to the slow-path flow of HEADER B 98, which also puts all the data into the header buffer or, if the header buffer is too small, uses a large (2K) host buffer for all the data. This means that on the unsolicited receive path, the host will only see either a header buffer or a header and at most, one data buffer. Note that data is never split between a header and a data buffer.

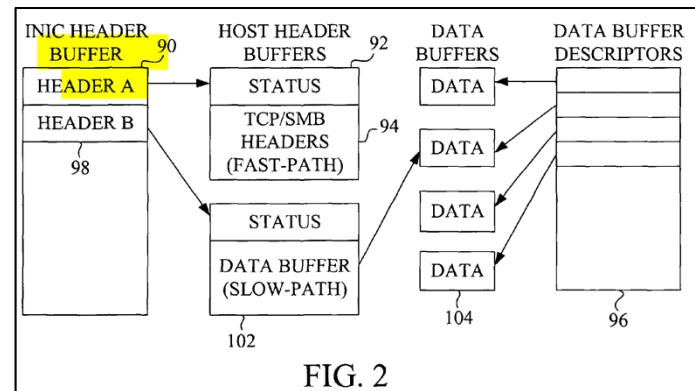
Ex. 2025.049 (880 App. Pub.) at ¶ [0115].

storing said header portion in a header buffer; and

Ex. 2025 (880 App. Pub.) at cl. 33.

[0271] Incoming packets delivered to ATCP only (not accepted by MSTCP) include TCP, TTCP or SPX packets destined for one of our IP addresses. This includes both slow-path frames and fast-path frames. In the slow-path case, the TCP frames are given in their entirety (headers included). In the fast-path case, the ATKReceivePacket is given a header buffer that contains status information and data with no headers (except those above TCP).

Ex. 2025.059 (880 App. Pub.) at ¶ [0271].



Ex. 2025.002 (880 App. Pub.) at Fig. 2.

See Paper 38 (1409 Opp. to Motion to Amend) at 5-6; Paper 50 (1409 Sur-Reply to Motion to Amend) at 4-5; Paper 38 (1410 Opp. to Motion to Amend) at 4-5; Paper 50 (1410 Sur-Reply to Motion to Amend) at 4-5.

PO in its reply relies entirely on new evidence for support

storing said header portion in a header buffer, wherein the header buffer is separate from said packet memory;

See, e.g., Ex. 2025 at Abstract, ¶¶ [0115], [0271], Fig. 2, Cl. 33.

Paper 20 (1410 Motion to Amend), App'x A at ii.

The '237 Application supports the limitation of storing the data portion of a received packet in a re-assembly buffer (such as a data buffer) and the header portion in a header buffer (such as an SRAM header buffer) that is separate from the packet memory (such as a DRAM frame buffer). *See, e.g., Ex. 2025, Fig. 2* (showing headers in header buffers and data in data buffers); ¶ [0593] (“As frames are received by the INIC from a network, they are *placed into 2K-byte DRAM buffers* Receive frame processing involves extracting this pointer from the Receive hardware queue, and setting up a DMA into an *SRAM header buffer* of the first X bytes from the *DRAM frame buffer*.”). The '809 Provisional also supports this limitation. *See, e.g., Ex. 2019 at 17* (Figure showing headers in header buffers and data in data buffers); 53 (“automatic movement of input frames into DRAM”); 57 (“The first step in receive processing is to dma the frame header into an SRAM header buffer.”). Ex. 2305 at ¶ 31. A POSA would understand that a SRAM is a different type of memory from a DRAM and, thus, the SRAM header buffer is separate from the DRAM frame buffer.

Paper 43 (1410 Motion to Amend Reply) at 1-2;
See Paper 50 (1409 Sur-Reply to Motion to Amend) at 4-5;
Paper 50 (1410 Sur-Reply to Motion to Amend) at 4-5.

Amended limitations are not identical to the original, as-filed claims

- “wherein the header buffer is ***separate from*** the packet memory” (claim 61)
- “wherein the header buffer is ***separate from*** the packet memory” (claim 79)
- “wherein the header buffer is ***separate from*** said packet memory” (claim 85)
- “wherein the header buffer is ***separate from*** the memory” (claim 87)

880 Patent: Disputes

4. Motions to Amend 880 Patents should be denied

- a) PO has not met its burden of production under 35 U.S.C. § 316(d) due to its failure to provide adequate written description support
- b) **The substitute claims are indefinite**
- c) The prior art combinations disclose the limitations of the substitute claims

A POSA would not know what “separate from” means in this context

- “... could mean that the header buffer and packet memory are located on ***the same memory device, but the physical location*** on the memory device where the header is stored ***is different*** from the physical location on the memory device whether the packets are stored”
- “... could refer to the memory device itself, such that the header buffer is on ***a different memory device*** than the packet memory”
- “... could mean that ***the virtual address for the header is separate from the virtual address for the packet***”

Ex. 1210.010 (1409 Lin Opp. Decl.) at ¶ 24; See also Paper 38 (1409 Opp. to Mot. to Amend) at 9-11; Paper 50 (1409 Sur-Reply to Mot. to Amend) at 7-8; Paper 38 (1410 Opp. to Mot. to Amend) at 8-10, Ex. 1210.010-.011 (1410 Lin Opp. Decl.) at ¶ 24, Paper 50 (1410 Sur-Reply to Mot. to Amend) at 7-8.

880 Patent: Disputes

4. Motions to Amend 880 Patents should be denied

- c) The prior art combinations disclose the limitations of the substitute claims
 - i. **“storing said header portion in a header buffer, wherein the header buffer is separate from the packet memory” (substitute claims 61, 79, 85, 87)**
 - ii. “re-assembling [said/a/the] data portion” / “re-assembler” (substitute claims 61, 79, 85, 87)

Exemplary proposed claim 61

61. (proposed substitute for claim 1) A method of transferring a packet to a host computer system, wherein the packet is received at a communication device from a network, comprising:

storing a first packet received at a network interface for the host computer system in a packet memory;

parsing a header portion of [[a]] said first packet received at [[a]] the network interface for the host computer system to determine if said first packet conforms to a TCP protocol;

generating a flow key to identify a first communication flow that includes said first packet, wherein said flow key includes a TCP connection for the communication flow;

associating an operation code with said first packet, wherein said operation code indicates a status of said first packet, including whether said packet is a candidate for transfer to the host computer system that avoids processing said header portion by the host computer system in accordance with said TCP protocol;

and

if said first packet conforms to the TCP protocol:

storing a data portion of said first packet in a re-assembly buffer;

storing said header portion in a header buffer, wherein the header buffer is separate from the packet memory;

re-assembling said data portion of said first packet with a data portion of a second packet in the communication flow; and

processing, by the network interface, said first and second packets according to the TCP connection, including updating a control block representing the TCP connection on the network interface.

Paper 20 (1409 Motion to Amend), App'x C at xix.

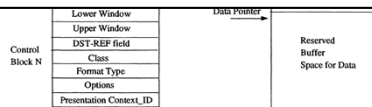
The “slower external memory” is a “packet memory” as claimed

234

Part Five Protocols



retransmission strategy was used. For a large window, the on-chip buffer may not be sufficient to hold the unacknowledged data packets for retransmission or to buffer data packets for resequencing, and slower external memory would be needed.



Host Tag : This tag is set on receipt of a host command, e.g. BYPASS_START, BYPASS_DMA, BYPASS_SYNC or BYPASS_RESTART.

STATUS : Indicates the status of the buffer, e.g. EMPTY, FILLING, FILLED or CLOSED.

Figure 4 Organization of internal bypass chip memory
 easily because the sequence number matches that of a previously received TPDU. At the sender end, if timer T1 expires, the transport entity can retransmit either the first TPDU, or all TPDU's (Go-back-N) waiting for acknowledgment. In this design the Go-back-N retransmission strategy was used. For a large window, the on-chip buffer may not be sufficient to hold the unacknowledged data packets for retransmission or to buffer data packets for resequencing, and slower external memory would be needed.

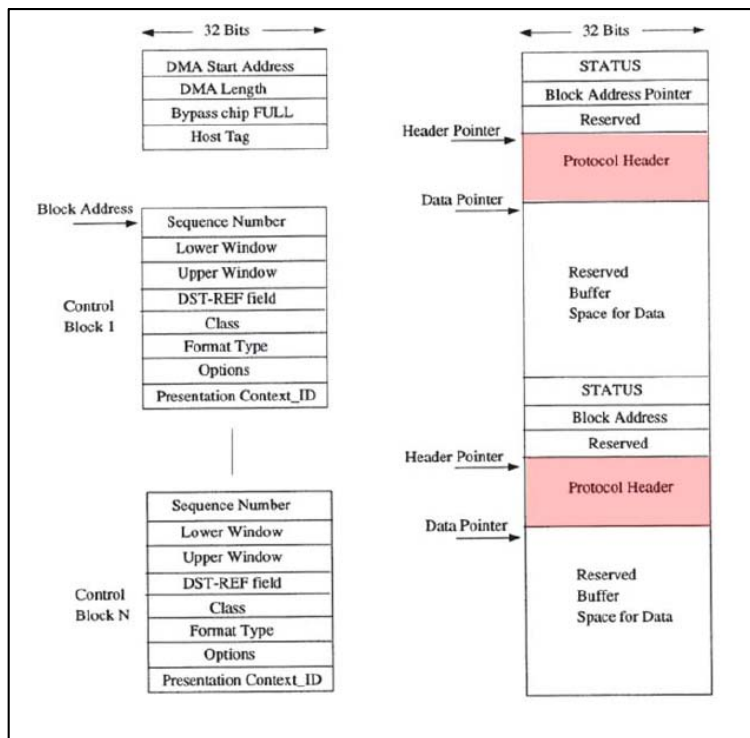
INTEL Ex.1015.011

Ex. 1015.011 (Thia);

See Paper 38 (1409 Opp. to Motion to Amend) at 13-14, 20-22;
 Ex. 1210.025, .053 (1409 Lin Opp. Decl.); Paper 50 (1409 Sur-Reply to Motion to Amend) at 9-10; Paper 38 (1410 Opp. to Motion to Amend) at 12, 16-17; Ex. 1210.045, .055 (1410 Lin Opp. Decl.); Paper 50 (1410 Sur-Reply to Motion to Amend) at 9-10.

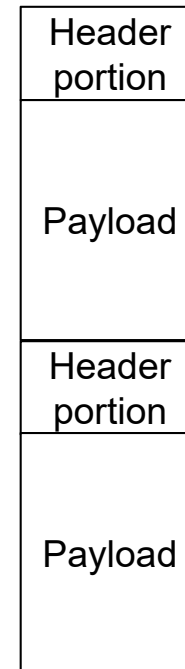
“header buffer” in internal memory is separate from external “packet memory”

Thia’s Internal Dual-Ported Memory Comprising Header Buffers



Thia’s External Packet Memory

Move for bypass processing



Ex. 1210.052-.053 (1409 Lin Opp. Decl.) (excerpting Ex. 1015 (Thia) at .011);

See also Paper 38 (1409 Opp. to Motion to Amend) at 22-23; Paper 50 (1409 Sur-Reply to Motion to Amend) at 10-11; Paper 38 (1410 Opp. to Motion to Amend) at 17; Ex. 1210.054-.055 (1410 Lin Opp. Decl.); Paper 50 (1410 Sur-Reply to Motion to Amend) at 10-11.

PO's (and Its expert's) rebuttal of petitioner's obviousness argument is based on a incorrect premise

header and data portions. Ex. 1015 at 011; Ex. 2305 at ¶ 23. As for the “slow packet memory,” This explains it is only used to store packets that are too large for the packet buffer on the ROPE chip. Ex. 1015 at 011 (“For a large window, the on-chip buffer may not be sufficient to . . . buffer data packets for resequencing, and slower external memory would be needed.”); Ex. 2305 at ¶ 24. Thus a received packet is

Paper 43 (1410 Motion to Amend Reply) at 8;
See Paper 50 (1409 Sur-Reply to Motion to Amend) at 9-11; Paper 50 (1410
Sur-Reply to Motion to Amend) at 9-11.

A “window” refers to the number of bytes, not the size of packets, that can be received

Flow control in TCP is handled using a variable-size sliding window. The *Window* field tells how many bytes may be sent starting at the byte acknowledged. A *Window* field of 0 is legal and says that the bytes up to and including *Acknowledgement number* – 1 have been received, but that the receiver is currently badly in need of a rest and would like no more data for the moment, thank you. Permission to send can be granted later by sending a segment with the same *Acknowledgement number* and a nonzero *Window* field.

Ex. 1006.545 (Tanenbaum96);
See Paper 50 (1409 Sur-reply to Motion to Amend) at 9-11;
Ex. 1210.056-.057 (1409 Lin Opp. Decl.);
Paper 50 (1410 Sur-reply to Motion to Amend) at 9-11;
Ex. 1210.058-.059 (1410 Lin Opp. Decl.).

PO's expert agrees that a window does not refer to size of packets

Q. Does – is the window referencing a TCP window?

A. No.

Q. What is it referencing?

A. Within the GO-back-N retransmission strategy, there is a window size. And so it's referencing that window size. And in that instance, it's referencing **how much data** can be buffered on the receive side....

Ex. 1254 (Almeroth Depo.) at 100:15-22;
See Paper 50 (1409 Sur-reply to Motion to Amend) at 9-10;
Paper 50 (1410 Sur-reply to Motion to Amend) at 9-10.

PO's expert disagrees that Thia's external memory is only for packets that are too large for internal memory

- Q. Is it your understanding that slower external memory could only be needed if the packets were larger than the on-chip buffer?
- A. I don't think I would agree with the "only" characterization. As the first part of the sentence says, it says, "The on-chip buffer may not be sufficient to hold the unacknowledged data packets for retransmission." I think there's a variety of scenarios under which that might be the case....

Ex. 1254 (Almeroth, Depo.) at 105:24-106:8;
See Paper 50 (1409 Sur-reply to Motion to Amend) at 9-10;
Paper 50 (1410 Sur-reply to Motion to Amend) at 9-10.

PO's rebuttal is based on a faulty premise contradicted by its expert

Q. So if you had a situation where the large window encompassed, let's just say ten packets, for example, if you had a case where the large window encompassed ten packets, and together, those ten packets were bigger than the on-chip buffer, that would be another circumstance where the slower external memory would be needed, right?

A. If your hypothetical asks me to assume that the ten packets are larger than what can be stored in the on-chip buffer, then I would agree that the slower external memory would be needed. I think that pretty much reads straight from the sentences we have been looking at on Page 11.

Q. And just to be clear, that's not the individual packets are too large to store, but together, the ten packets are too large to store, right?

[objection omitted]

A. If that's part of your hypothetical, then I think that's fine.

Ex. 1254 (Almeroth Depo.) at 107:14-108:8;
See Paper 50 (1409 Sur-reply to Motion to Amend) at 9-10; Paper
50 (1410 Sur-reply to Motion to Amend) at 9-10.

880 Patent: Disputes

4. Motions to Amend 880 Patents should be denied

- c) The prior art combinations disclose the limitations of the substitute claims
 - i. “storing said header portion in a header buffer, wherein the header buffer is separate from the packet memory” (substitute claims 61, 79, 85, 87)
 - ii. “re-assembling [said/a/the] data portion” / “re-assembler” (substitute claims 61, 79, 85, 87)

PO's arguments on re-assembly/re- assembler are not new and are similarly wrong

Thia and Tanenbaum96 also fail to teach re-assembling data portions of two data packets in the communication flow. In particular, Thia's bypass chip does not perform any reassembly of the PDUs into larger data blocks. Ex. 1015 at 014 ("There is no segmentation/reassembly within the bypass path"). In other words, Thia's bypass chip performs some of the protocol processing functions (such as validating checksums, decoding headers, etc.) but then provides the PDU to the host for reassembly into a larger data block. Ex. 2305 at ¶ 28. Tanenbaum96 also does not disclose re-assembling data portions of two data packets in the same communication flow. *Id.* at ¶ 29.

Paper 43 (1410 Motion to Amend Reply) at 10.

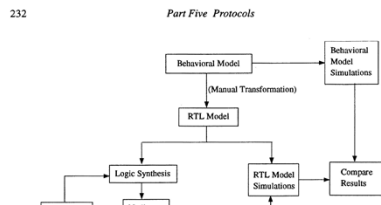
28. It is also my opinion that Thia and Tanenbaum also fail to disclose re-assembly of packet data. In particular, Thia's ROPE chip does not perform any reassembly of PDUs into large data blocks. Indeed, Thia explains at page 14 that Alacritech Exhibit 2305, Page 12 "[t]here is no segmentation/reassembly within the bypass path" In other words, Thia's ROPE chip performs some of the protocol processing functions (such as validating checksums, decoding headers, and the like) but then provides the entire PDU to the host for reassembly into a larger data block.

29. Additionally, Tanenbaum does not disclose the NIC re-assembling data portions of two packets in the same communication flow.

Ex. 2305.012 (Almeroth Decl. ISO Reply) at ¶¶ 28-29.

See Paper 50 (1409 Sur-reply to Motion to Amend) at 11-12;
Paper 50 (1410 Sur-reply to Motion to Amend) at 11-12.

The evidence PO and its expert rely on is for transmitting, not receiving



- 2) For subsequent bypassable packets, the host processor initiates the BYPASS_DMA procedure which checks for free buffer space in the bypass chip and programs the DMA by sending the starting address pointer where the PDU is located, and its total length. The destination address is supplied by the bypass chip. Arbitration for the host processor bus between the host and DMA is provided by the DMAreq and DMAack lines. DMA transfers the PDU into the internal dual-ported SRAM (Static RAM). Buffers are pre-allocated in fixed sizes and are accessed by a simple round robin scheme using a set of buffer pointers.

INTEL Ex.1015.009

Ex. 1015.009 (Thia).

See Paper 50 (1409 Sur-reply to Motion to Amend) at 11-12;
Paper 50 (1410 Sur-reply to Motion to Amend) at 11-12.