High Speed Communication for Distributed Applications

Alfred C. Weaver

Computer Science Report No. TR-93-10
January 18, 1993

HIGH SPEED COMMUNICATION FOR DISTRIBUTED APPLICATIONS

Alfred C. Weaver, Director
Computer Networks Laboratory
Department of Computer Science
University of Virginia
Charlottesville, Virginia 22903 U.S.A.

Abstract: The distributed applications of the 1990s, especially those related to factory automation, will require a new set of communications services and will demand a new standard of performance. In this paper we discuss the design of the *Xpress Transfer Protocol* which is intended to provide these new services. We explain its history and motivation as well as its features and functionality. In addition, we describe a new technology for implementing communications protocols in hardware—the *Protocol Engine*. The PE provides an end-to-end silicon path between communicants across an intervening network. In its first implementation, the PE will process network traffic at the 100 Mbits/sec rate of a commercial FDDI local area network.

Background

An international group of researchers, led by Greg Chesson at Silicon Graphics Inc. (Mountain View, California), has developed a new *transfer* layer communications protocol (the integration of the *transport* and *network* layers from the OSI Reference Model) specifically designed to meet the needs of modern, distributed computing systems. The *Xpress Transfer Protocol* (XTP) [1] represents the synthesis of the best ideas developed by several extant protocols and then extends those ideas with new services designed to support distributed applications and embedded real-time systems. As a result, XTP offers a unique combination of services, including fast connection setup, large data pipeline size, selective acknowledgement, selective retransmission, flow, error, rate and burst control, message priority, intraprotocol scheduling, reliable and unreliable datagrams, out-of-band data delivery, multiple addressing formats (including Internet and ISO 8348 Network layer addresses) and reliable transport multicast. XTP's design reflects a clear separation of policy from mechanism—XTP provides multiple communications mechanisms, but XTP's user is allowed to select those mechanisms which support the user's desired communications paradigm.

XTP's new services should prove especially valuable for the types of applications we expect to see in the 1990s: communications between computation engines and high-resolution displays; remote medical imaging; transaction-based client/server systems; transmission of terabit databases; real-time control systems for ships, aircraft, space vehicles, and factories; and multimedia teleconferencing. As an example of XTP's emerging acceptance, the United States Navy has selected XTP as one of the protocols in Safenet (Survivable Adaptable Fiber Optic Embedded Network) [2], the Navy's next-generation specification for mission critical communications systems.

Another distinguishing feature of XTP is that it is specifically designed for implementation in hardware as the *Protocol Engine*. When coupled with a commercial FDDI network, the PE will be able to process packets at the same rate as their arrival from the network medium. Thus we foresee providing transport and network layer services at a rate of 100 Mbits/sec in the near future, and at gigabit/sec speeds as the hardware technology matures. The commercialization of the Protocol Engine, the VLSI realization of XTP in silicon, and the adoption of XTP by the Navy's Safenet community all point to a new era of high throughput, low latency communications service for distributed real-time systems.

In this paper we look first at the history and motivation of XTP, then at its new functionality, and finally at the Protocol Engine design which will enable XTP to establish an end-to-end "silicon data path" between applications.

Motivation for XTP

The fundamental nature of computing and control systems is changing. No longer confined to an individual processor, applications are distributed across a multiplicity of computers, thereby making the communications network a critical factor in the overall systems architecture. For a new application, and in particular a factory control application, the choice of the communications protocol controls the functionality of the services offered as well as the ultimate performance observed by the service users.

To be advantageous, any new protocol for distributed systems must provide more and better services than are currently available. For example, typical transport protocol provide either a totally reliable service (connection-oriented) or an unacknowledged service (connection-less), and nothing in between. A better strategy would be to provide a variety of mechanisms and let the user select which communications paradigm best suits his particular application. XTP addresses this issue by supporting transactions, user-defined service reliability, multipeer communication (multicast), and latency control.

On the performance side, any new protocol should permit a variety of implementations—from all software to all hardware—so that system designers can balance performance with economics. While the protocol itself must remain invariant, its implementation should scale gracefully from megabits to gigabits, and likewise should span LANs and MANs and WANs without alteration.

Why Change?

The networking community has designed a number of successful transport protocols, of which the best known are the Transmission Control Protocol (TCP) [3] and the ISO Transport Protocol class 4 (TP4) [4]. Since these are in everyday use, why change?

The motivation for change comes about because the networking world has changed enormously in the past decade. Whereas TCP and TP4 were designed in an era when bit error dates on the medium were typically 10^{-5}, modern fiber optic networks routinely provide bit error rates of 10^{-12} or less; clearly this affects the retransmission strategy. Similarly, older protocol were designed when transmission speeds were slow (e.g., 56 Kbits/sec over a dedicated telephone line). As transmission speed increases, from the 10 Mbits/sec of Ethernet to the 100 Mbits/sec of FDDI to the gigabit/sec speeds of tomorrow's ATM (Asynchronous Transfer Mode) networks, the end-to-end bitpipe is more densely packed than ever before. At these high speeds, new mechanisms for error control and error avoidance will be particularly valuable (see [5]).

Perhaps most compelling is the observation is that the user of the network has changed. Rather than simply being a vehicle for the exchange of data between two peers in a fully reliable, connection-oriented fashion, the network can now be viewed as a *virtual backplane* for distributed applications. Especially in the real-time control environment of the modern factory, networks must support synchronization, transaction processing, multiple priority messages, reliable one-to-many transmission, and both reliable and unacknowledged datagrams, in addition to the traditional connection-oriented services.

New Functionality in XTP

A complete description of XTP's functionality may be found in [6]. This section summarizes the most important design issues.

Multicast. Of all of XTP's features, multicast may be the most innovative and important. XTP allows a user to define a *transport* multicast group; thereafter, transmissions from the multicast transmitter are routed to all members of the multicast receiver group. Data transmission is reliable; that is, receivers may NACK missing data, causing it to be retransmitted transparently. Multicast is also *progressive*, in that the failure of a multicast receiver does not impede the progress of the group. Although multicast group management is not a part of the XTP specification, our research group has made a proposal about how to manage the reliability of the multicast group membership in [7,8].

Priorities. A well-known problem with traditional protocols is that they are not very responsive to user data of varying importance. This is particularly crucial for real-time systems. In the same way that FDDI discriminates among data classes (synchronous vs. asynchronous) at the datalink layer, XTP provides a discrimination mechanism (called SORT) at the transport layer. Users may optionally encode a 32-bit SORT value to indicate a message's relative priority. Within the XTP delivery sub-system, including both end-systems and all intermediate routers, the SORT value of incoming packets is used to influence intraprotocol scheduling. At each transmission opportunity, a node selects its most important packet and processes it. While not as responsive as a pre-emptive scheme, it is more efficient, and provides discrimination among competing packets with a granularity of one packet's transmission time, thus drastically reducing priority inversion.

Rate and Burst Control. Every transport protocol provides control mechanisms for bit error detection (e.g., checksums) and for flow control (buffer management). However, as characteristic network error rates drop from, say, 10^{-5} on copper media to more like 10^{-12} on fiber optic media, bit error rate is much less of a problem; on fiber optic LANs such as FDDI, for example, data loss from buffer overruns is much, much more of an issue than data corruption due to bit errors.

XTP thus introduces *rate* and *burst* control. Rate control allows the receiver to specify a maximum rate (bytes/sec) at which the transmitter should emit data; this helps synchronize not only the transmitter and receiver, but, properly used, the end-systems as well. Burst control allows the receiver to specify the maximum amount of data (bytes/burst) which the receiver can handle effectively; this augments flow control in avoiding buffer starvation. In addition, routers may participate in rate and burst control, so that a connection can be optimized not only for the end-system but also for the route being used.

Address Translation. Connection establishment ultimately requires the resolution of a network address to a connection identifier. Since sending large addresses repeatedly (e.g., in every packet) reduces network efficiency, some protocols utilize a form of address translation such that, once a connection is established, subsequent packets need not carry a full network address. The efficiency of the translation scheme is therefore a concern, especially for routers. Address lookups can be accelerated with cacheing schemes, and Van Jacobson's header prediction technique for TCP is an example of a helpful scheme. Header prediction has less impact if the traffic pattern is random (e.g., an X-Window server), and increasing the header cache size to accommodate a large number of active connections can increase its overhead to nearly that of the original address translation problem.

To counteract this effect, XTP evolved into a KEY-based scheme. The initial XTP packet which sets up an XTP connection (called a FIRST packet) contains an *address segment* and an *address type* field. Through a hardware hashing function, initial addresses are resolved into a 32-bit KEY field. Thereafter, subsequent packets can be identified by a table lookup on their KEY field. Note that XTP does not introduce (yet another) network addressing plan; XTP can operate with IP addresses, or ISO 8348 Network layer addresses, or any other network addressing plan.

Retransmission. Error detection results in retransmission for connection-oriented protocols. TCP and TP4 both use "go-back-n" in which the transmission scheme is reset and retransmission begins with the lost byte or packet. This scheme may redeliver information which has previously been sent and received correctly. XTP implements a more flexible mechanism. Receivers can provide a vector that indicates what *spans* of information have been correctly received. Using selective retransmission, only the missing data need be retransmitted. Selective retransmission is an option, not a requirement, and so "go-back-n" may still be used if appropriate.

Acknowledgement Control. TCP and TP4 provide automatic acknowledgements for packets received correctly. This again reflects the designer's underlying assumption that the network often loses packets and that positive acknowledgement is therefore a virtue. This decision embeds policy with mechanism. XTP allows the user to decide whether and when acknowledgements are desirable. Acknowledgements are requested by the transmitter by setting a status request bit, thus separating policy from mechanism.

This reflects another general philosophy in XTP: the transmitter is in control. Receivers do what they are asked to do, when they are asked to do it, but in general they do not automatically provide information which is not requested.

Out-of-band Data. A common protocol problem is how to send information from one user to another without embedding it in the data stream itself (i.e, out-of-band data). This is useful for passing application-layer control information about the state of the end-user processes, or possibly for passing application-layer semantic information regarding the data stream itself. XTP accomplishes out-of-band transmission by permitting each packet to carry eight bytes of *tagged* data. If used, XTP delivers the tagged data to the user, along with a flag indicating its presence, but XTP itself never interprets the data. A potential use for tagged data is to timestamp time-critical data.

Data Pipeline Size. How much data can be outstanding on any one connection? The answer depends upon the design of the sliding window protocol. If the size of the sliding window is too small, then the protocol will revert to stop-and-wait operation when used with high capacity networks. XTP anticipated this environment and provides a 32-bit sequence number and sliding window (i.e., a four gigabyte sequence space). The sequence number can be enlarged easily to 64 bits to accommodate future terabit/sec networks.

Connection Services. For reasons of reliability, some communications are connection-oriented, and thus the efficiency of building and maintaining connections is a concern. TP4, for instance, uses a six-packet handshake to transfer information reliably: two packets to request and acknowledge a connection, two packets to send and acknowledge data, and two packets to release

and confirm the close of the connection. XTP accomplishes the same reliability with a three-packet handshake.

Alignment. Data alignment may seem a simple notion, but its use pays big dividends. XTP aligns its major fields on 4-byte boundaries. Rather than actually aligning non-aligned data, which would involve copying, XTP uses an *offset* field and a *length* field; the former provides an offset from the beginning of the data frame to the beginning of user data while the latter allows the user to identify the last byte of user data in a frame which may be physically longer. Thus the transmitted data frames are always aligned, even if the data inside them is not. Since data copying is a very time consuming operation, mechanisms which seek to avoid data copying are seen as very advantageous.

Policy vs. Mechanism

XTP attempts to provide mechanisms, rather than to implement any particular pre-established policy; XTP realizes that only the user has sufficient information to truly optimize the parameters of a data exchange. To that end, XTP supports some special modes which the user may elect.

While some classes of transfer are adequately served by a try-as-best-you-can policy, others require more synchronization between transmitter and receiver to be effective. If, for example, it is pointless to transfer data unless it is known that the receiver has space for the data, that policy can be implemented using *reservation mode*. When reservation mode is selected, the transmitter forbids the user from including in its allocation any buffer space not dedicated to this association. This forces the transmitter and receiver to adopt a conservative flow control policy in which the transmitter always knows that there is buffer space available for every packet it sends.

Some data types should never be retransmitted. If, for example, subsequent data will soon be available, it may be better to drop an errant packet rather than retransmit stale data. This is accomplished using XTP's *noerror* mode, which defeats the normal retransmission scheme. Data received in error is marked as being in error, but no attempt is made to retransmit it. This preserves the sequentiality of data but permits gaps. An example would be sending successive frame buffers for a video display at a rate of, say, 30 frames/sec. If an individual frame is lost, the receiver could just ignore it and wait for the next one rather than attempt a repair.

Some data transfers need to distinguish between what the XTP receiver has received vs. what the receiving XTP user has received. To permit that distinction, the XTP transmitter may make two types of requests: SREQ (status request) and DREQ (delivered status request). By setting the SREQ or DREQ bits in the protocol control fields, the transmitter forces the receiver to emit a control packet containing the sequence number of the data byte last received by XTP (in response to SREQ) or of the data byte last delivered to the XTP user (in response to DREQ). These sequence numbers are returned as RSEQ (received sequence number) and DSEQ (delivered sequence number) in the control packet generated by the receiver.

What should be clear by now is that XTP offers extraordinary flexibility to the XTP user by providing mechanisms rather than implementing some pre-selected policy. This allows XTP to be dynamically responsive to the user for a wide range of applications.

The Protocol Engine

The goal of the Protocol Engine (PE) is to process information at the same rate that it arrives from the network medium. This so-called real-time packet processing requires a *flow through* architecture in which data is moved, managed, and manipulated such that one packet can be processed before a second packet has fully arrived. This in turn requires a pipelined design in which the various protocol operations and data movements can be overlapped and executed in parallel.

Figure 1 shows the dataflow required when the PE is connected to a commercial FDDI network. At the 100 Mbits/sec data rate of FDDI, the arrival of two back-to-back 60-byte packets permits only 5 microseconds for the processing of each packet. The PE accomplishes this goal by pipelining the protocol parsing, address parsing and lookup, packet steering, forwarding, and checksum calculation. These operations are assigned to a short pipeline of parallel RISC engines which accomplish the operations required at each stage of the pipe.

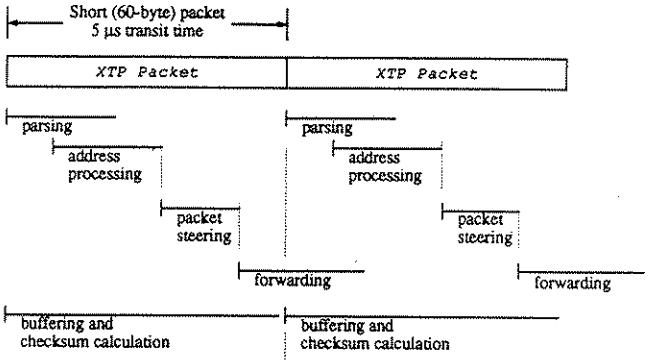

Figure 1.
Packet Pipelining in the Protocol Engine

To accomplish the goal, four VLSI chips are currently being designed to handle the various stages of the pipeline. The chips are the MPORT (MAC Port), which interfaces to the underlying network; the HPORT (Host Port), which interfaces to the host's backplane bus; the BCTL (Buffer Controller), which provides temporary storage between the MPORT and HPORT; and the CP (Control Processor), a commercial RISC processor that executes some of the more complicated protocol operations. Figure 2 shows a typical configuration.

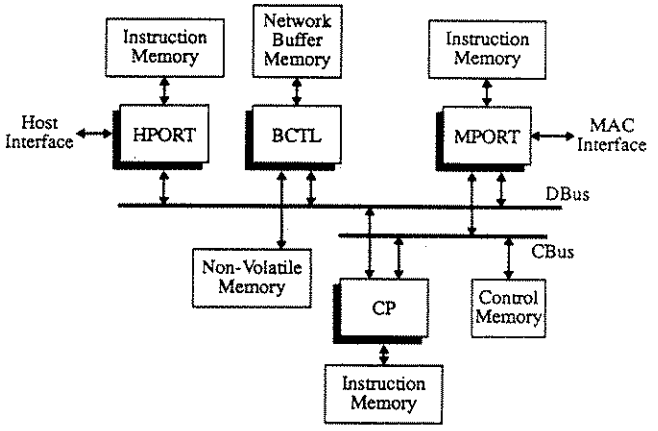

Figure 2.
Protocol Engine Block Diagram

MPORT. The MAC Port chip moves data between the network and the buffer memory. In its first version, the selected MAC will be a commercial FDDI chipset. Figure 3 shows a block diagram of the MPORT.

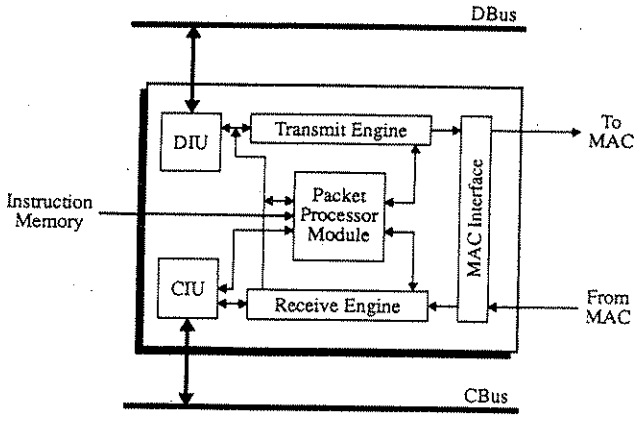

Figure 3.
MPORT Block Diagram

On its receive side, the MPORT's Receive Engine accepts data from the MAC and performs two protocol-specific hardware checksums on the packet. The first is the so-called header checksum, which protects the addressing field of the packet. If this check fails the packet is discarded, since the Transport Service Access Point would be in doubt. The second check is the checksum across the data field of the packet.

As part of XTP's capability to support user-specified levels of reliability, a failed data checksum does not automatically force the packet to be discarded. If the packet is in error, it is marked as being in error and it is an XTP protocol option as to whether it should be discarded or delivered (complete with the notation that data errors have been discovered). The Packet Processor Module parses the protocol header, extracts protocol state information, demultiplexes the packet according to its context (i.e., its intended end-to-end association), and steers the data to the Buffer Controller. On its transmit side, the Transmit Engine moves data from the Data Bus Interface Unit (DIU) to the transmit side of the MAC.

HPORT. The Host Port provides high speed Direct Memory Access (DMA) transfers between the host's backplane bus, its own internal FIFOs, and the Network Buffer Memory. It packetizes data supplied by the host and calculates transport checksums for each packet. Figure 4 shows its block diagram.

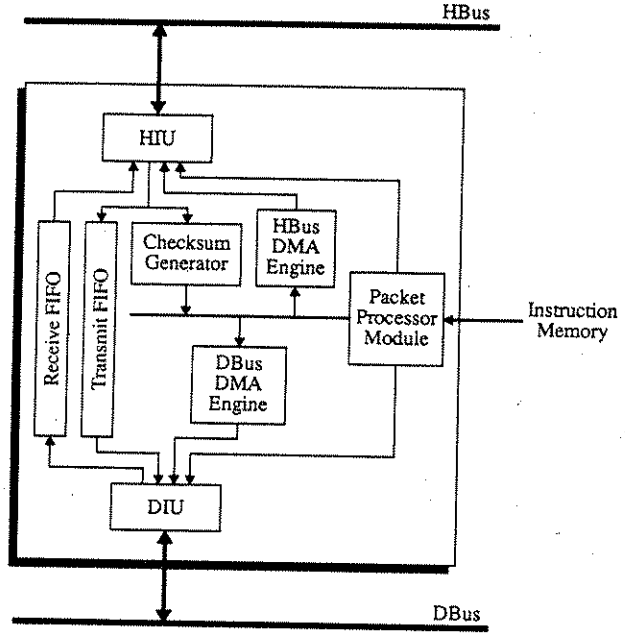

Figure 4.
HPORT Block Diagram

As data flows toward the network, the HPORT generates the packet header and trailer, computes and inserts the checksum, inserts the sequence number, and performs the utility functions of byte swapping, word alignment, and padding. As data flows to the user, the HPORT separates the protocol control data from the user data and delivers both to separate areas of user memory.

BCTL. The Buffer Controller arbitrates access to the Data Bus (DBus) and provides an interface for data transfer from the DBus to either the Network Buffer Memory or the Non-Volatile Memory. The BCTL's block diagram is shown in Figure 5.

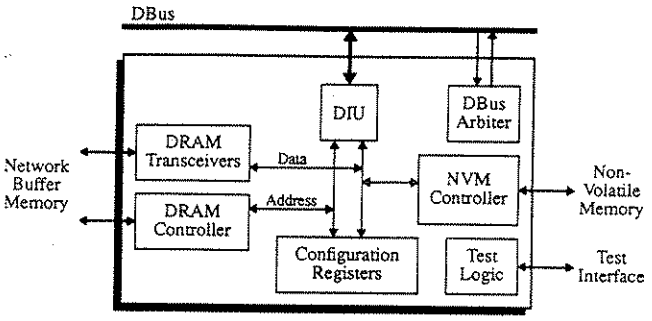

Figure 5.
BCTL Block Diagram

The DBus Arbiter controls mastery of the DBus among the three DIUs in the HPORT, MPORT, and BCTL. Data destined (temporarily) for the Network Buffer Memory arrives on the DBus, flows through the DRAM transceivers, and out to the DRAM itself. The DRAM controller provides the necessary read, write, and refresh operations. The Non-Volatile Memory Controller interfaces to an EEPROM which stores the microcode for the HPORT and MPORT.

CP. The Control Processor provides high-level protocol processing functions such as association management and path management. While the MPORT, HPORT, and BCTL provide generic packet processing functions, the CP accomplishes protocol-specific actions. The CP is expected to be a commercial, microprogrammable RISC processor.

The three-chip set of MPORT, HPORT, and BCTL form a *protocol accelerator*; adding the CP makes it a *protocol engine*. The protocol accelerator is expected to be available as a commercial product from Protocol Engines Inc. in late 1992 or early 1993 [9].

Off-Host Processing

The intent of the protocol engine is to move network system processing off-host. This means much more than protocol processing, which accounts for only a portion of the end-to-end latency encountered in a typical communication. The protocol engine approach addresses data movement, buffer management, interfaces to the operating system and application program, physical interfaces to the network and the backplane bus, checksum calculation, and device driver considerations.

One of the protocol engine's strengths is its use of parallelism—in the system block diagram shown in Figure 2 above, the host interface, the network interface, protocol parsing, and internal buffer control are essentially parallel processes. Although any particular packet is handled serially, the processes themselves operate in parallel, resulting in an effective processing pipeline between network and host.

Of course, off-host processing is not a guaranteed win. There are a number of other considerations which affect overall performance. Among them are: speed of the backplane bus, contention for bus cycles, number of data copies made while moving

information between the host and the protocol processor, number of hardware interrupts generated, choice of a shared memory versus a message passing architecture between host and front-end, and the number and frequency of interactions with the host's operating system. However, given competent design and implementation, the protocol engine should prove to be a substantial asset for moving network subsystem processing off-host.

Experience

The Computer Networks Laboratory at the University of Virginia has had much experience with developing XTP for several different hosts (Intel 80x86, Motorola 680x0, SUN, RS/6000), for several different operating systems (pSOS, pSOS+, VRTX, ZEK, DOS), and for several different networks (Ethernet, token ring, FDDI). Our experiences are further described in [10-15]. Other groups, particularly those in Europe, have also documented their experiences in [16-19].

Summary

XTP is a new venture in transport and network protocol design. When implemented in software, XTP provides new protocol functionalities (e.g., multicast, priorities) that are expected to be of significant benefit to the next generation of modern distributed applications. When implemented in hardware in the form of the *Protocol Engine*, XTP should off-load much of the host's network subsystem processing, thereby increasing network throughput and reducing end-to-end latency.

Acknowledgements

We gratefully acknowledge the technical and financial support of our research sponsors: Naval Ocean Systems Center, Naval Surface Warfare Center, Office of Naval Research, Sperry Marine, IBM, E-Systems, SAIC, Protocol Engines Inc., and the Virginia Center for Innovative Technology. I am forever grateful to the staff (past and present) of the Computer Networks Laboratory at the University of Virginia, and especially acknowledge the contributions of four individuals: Robert Simoncic and John Fenton, who are the primary architects and implementors of our XTP systems, and Tim Strayer and Bert Dempsey, who have authored many of our publications regarding the design of high speed protocols.

References

[1] Xpress Transfer Protocol, version 3.6, Protocol Engines Inc., 1900 State Street, Suite D, Santa Barbara, California 93101 USA, January 11, 1992.

[2] Survivable Adaptable Fiber Optic Embedded Network II (Safenet II), Military Handbook, January 10, 1992.

[3] Postel, J., ed., "Internet Protocol - DARPA Internet Program Protocol Specification," RFC 791, USC/Information Sciences Institute, September 1981.

[4] International Organization for Standardization, "Information Processing Systems - Open Systems Interconnection - Transport Protocol Specification, Draft International Standard 8073, July 1986.

[5] Kleinrock, Leonard, "The Latency/Bandwidth Tradeoff in Gigabit Networks," *IEEE Communications Magazine*, April 1992, pp. 36-41.

[6] Strayer, W.T., Dempsey, B.J., and Weaver, A.C., *XTP: The Xpress Transfer Protocol*, Addison-Wesley, July 1992.

[7] Dempsey, Bert J., "Design and Analysis of a Transport Layer Reliable Multicast," M.S. thesis, Department of Computer Science, University of Virginia, Charlottesville, Virginia, January 1991.

[8] Dempsey, B.J., Fenton, J.C., and Weaver, A.C., "The Multidriver: A Reliable Multicast Service for the Xpress Transfer Protocol," *15th Local Computer Networks Conference*, Minneapolis, Minnesota, October 1-3, 1990.

[9] "The PE1000 Protocol Engine Chipset," Protocol Engines Inc., 1900 State Street, Suite D, Santa Barbara, California 93101, USA.

[10] Weaver, A.C., "The Xpress Transfer Protocol: A Fast Transport Protocol for High Speed Networks," *International Workshop on Advanced Communications and Applications for High Speed Networks*, Munich, Germany, March 16-19, 1992.

[11] Strayer, W.T., and Weaver, A.C., "Is XTP Suitable for Real-Time Distributed Systems?", *International Workshop on Advanced Communications and Applications for High Speed Networks*, Munich, Germany, March 16-19, 1992.

[12] Dempsey, B.J., and Weaver, A.C., "Adaptive Error Control for Multimedia Data Transfers," *International Workshop on Advanced Communications and Applications for High Speed Networks*, Munich, Germany, March 16-19, 1992.

[13] Weaver, A.C., "Making Transport Protocols Fast," *16th Local Computer Networks Conference*, Minneapolis, Minnesota, USA, October 13-17, 1991.

[14] Simoncic, R., Weaver, A.C., and Colvin, M.A., "Experience with the Xpress Transfer Protocol," *15th Local Computer Networks Conference*, Minneapolis, Minnesota, USA, October 13-17, 1991.

[15] Weaver, A. C., "Thoughts on Fast Protocols," *15th Local Computer Networks Conference*, Minneapolis, Minnesota, USA, October 13-17, 1991.

[16] Henrichs, Bernd, "XTP Specification and Parallel Implementation," *International Workshop on Advanced Communications and Applications for High Speed Networks*, Munich, Germany, March 16-19, 1992.

[17] Miloucheva, Ilka, and Rebensburg, Klaus, "XTP Service Classes and Routing Strategy for an Integrated Broadband Environment," *International Workshop on Advanced Communications and Applications for High Speed Networks*, Munich, Germany, March 16-19, 1992.

[18] Saulnier, Emilie T., and Mitchell, Robert J., "A Multi-Processor Partitioning of XTP," *International Workshop on Advanced Communications and Applications for High Speed Networks*, Munich, Germany, March 16-19, 1992.

[19] Braun, Torsten, Stiller, Burkhard, and Zitterbart, Martina, "XTP and VMTP on Multiprocessor Architectures," *International Workshop on Advanced Communications and Applications for High Speed Networks*, Munich, Germany, March 16-19, 1992.