XTP: A New Communications Protocol for Factory Automation

Alfred C. Weaver

Computer Science Report No. TR-93-09
January 18, 1993

IEEE/SICE International Workshop on Emerging Technologies
for Factory Automation -- State of the Art and Future Directions
Cairns, North Queensland, Australia
August 17-19, 1992

XTP: A New Communications Protocol for Factory Automation

Alfred C. Weaver

Computer Networks Laboratory, Department of Computer Science, University of Virginia, Charlottesville, Virginia 22903, U.S.A.

Abstract

As modern factory automation systems become more distributed through the use of local area network technology, their need for high throughput, low latency communication increases. The Xpress Transfer Protocol (XTP) provides a rich new set of functionalities specifically targeted to the needs of these distributed, real-time applications. When implemented in hardware with the *Protocol Engine*, XTP can provide transport layer communications services at the same speed as the underlying network media. This paper discusses the motivation and history of XTP development, the new functionality which the protocol provides, the protocol optimizations which enhance performance, and current plans for its VLSI implementation.

1. MOTIVATION AND HISTORY OF XTP

Traditional transport protocols were designed for an era when network bandwidth was low and error rates were high. Thus they made assumptions about where protocol processing should be performed (in the host computer), how data should be transmitted (connection-oriented), how to recover from errors (go-back-n retransmission), and how many applications should simultaneously communicate (two). However, the move toward distributed computer systems in general, and distributed factory automation environments in particular, negate these assumptions. Modern systems can benefit from moving the protocol processing off-host onto an attached communications processor (implemented in VLSI hardware). We can take advantage of this paradigm shift by defining a new communications protocol which, in addition to being conducive to silicon implementation, provides the new services required of distributed applications.

The movement toward off-host processing permits us to off-load more than just the protocol processing; indeed, protocol processing alone is just a fraction of the work required in a network interface. To really influence performance, we must optimize *network subsystem processing*, defined here to include multiple layers of protocol processing, data movement to and from the network, data movement to and from the host, buffer management, operating system interface, application program interface, backplane bus interface, and network device handling.

With that goal in mind, a group of international researchers, under the leadership of Dr. Greg Chesson at Silicon Graphics Inc., undertook the challenge of designing a new communications protocol for distributed applications which would provide the new functionality and higher performance required by this new environment. Corporate membership in the design group included AMD, Artel/NASA, Boeing, Concurrent, Crosfield Electronics, DY-4, E-Systems, IBM, Intel, Intergraph, Interphase, Linotype-Hell, Lockheed, SBE, Scitex, Unisys, and Xerox. In addition, a large number of not-for-profit organizations participated, including Admiralty Research Establishment, Bowman Gray School of Medicine, Concordia University, Electronics and Telecommunications Research Institute, Laboratoire de Genie Informatique, Johns Hopkins University, National Center for Supercomputer Applications, Naval Ocean Systems Center, Naval Surface Warfare Center, Superconducting Supercollider Lab, University of Kentucky, University of Massachusetts, University of Melbourne, and University of Virginia.

This group undertook the task of trying to understand what functionality and performance would be required by these new distributed applications. In addition to the traditional connection-oriented services provided by classic protocols such as the Transmission Control Protocol (TCP) [1] and the ISO Transport Protocol class 4 (TP4) [2], we saw a need for transactions (to support database query/response), reliable datagrams (for certain types of sensor data distribution), multicast (communication among any number of peers, rather than between just two peers), latency control and intra-protocol scheduling (to support real-time applications), and more flexible error control procedures (rather than the current choice of either fully reliable or totally unacknowledged). We think that modern factory automation systems will exploit all of this new functionality.

The performance requirements of individual systems are clearly application-dependent. Nevertheless, we saw a clear trend toward requiring sub-millisecond delivery of *transport* layer services (not just datalink layer services); for example, robotic arm control and parts placement control require reliable but fast data exchange between the controller and the controlled device.

One way to attack the problem was to make a list of changes required to some extant protocol, but after reflection, we concluded that we were more likely to succeed if we designed a new protocol rather than tinker with an old one. If this decision ultimately proves to be the correct one, then XTP will be accepted by the standardization bodies and eventually issued as an international standard; if we were in error in judging the inertia of the current protocol suites, then, at the least, XTP will provide some experience with new protocol mechanisms which should prove helpful in augmenting existing standards.

In promoting XTP, we make no religious arguments either for or against extant protocols. Rather than saying that XTP is *better* than protocol X in some absolute sense, we say that XTP is *different* and then explain how it is different and what advantage is expected to accrue from that difference. In that sense, the design of XTP has been a legitimate academic challenge, and our ultimate goals of standardization, commercialization, and acceptance are controlled more by the user community than by the designers.

Our philosophy throughout has been to reuse whatever protocol mechanisms have been shown by experience to be useful, and then to augment those ideas with new services and new mechanisms which we think will be required in the 1990s. Toward that end, XTP gratefully acknowledges the work which has gone before, particularly the lessons learned and experience gained with TCP/IP [1,3], TP4 [2], Delta-t [4], NETBLT [5], GAM-T-103 [6], VMTP [7], URP [8], and Datakit [9]. From these beginnings we have designed new services, provided a set of orthogonal protocol options, separated policy from mechanism, enabled fast implementation through intelligent design, and encouraged VLSI implementation. The documentation for the resulting protocol can be found in two places: the official XTP specification [10] and a professional reference book [11] which describes XTP's history, the protocols which influenced it, the XTP architectural model, XTP's protocol procedures, packet structure, and packet formats, the use of address encapsulation, the role of multicast, and the design of the Protocol Engine.

2. FUNCTIONAL ADVANTAGES

XTP is seen to have functional advantages with regard to accommodating a large data pipeline size, providing rate and burst control, managing a rich message priority structure, transporting out-of-band data, allowing selectable error control options, permitting selectable flow control options, separating policy from mechanism, and providing a transport layer reliable multicast facility. These functional advantages are discussed in turn.

Data Pipeline Size. As technology progresses, modern distributed systems will have more and more data outstanding in the data pipeline among communicating applications. This increased density in the bitpipe is unavoidable as networking technology is upgraded from the 10 Mbits/sec of Ethernet to the 100 Mbits/sec of FDDI to the gigabit/sec speeds of ATM networks. In anticipation of these ever-increasing speeds and the resulting increase in the amount of data outstanding on each connection, XTP currently supports a 32-bit sequence space which permits a 4 gigabyte sliding window. XTP can also use an internal synchronization field to augment its sequence number field, resulting in a 64-bit sequence number; this possibility will ease integration into future terabit/sec networks.

Error Prevention. All transport protocols protect the application from media errors (by using CRCs) and from protocol, host, or router errors (by using transport checksums). XTP provides these services as well. But in addition, XTP recognizes that, with the advent of fiber optic technology and its characteristic error rate of 10^{-12}, the probability of packet loss from buffer overflow and internal system congestion is far higher than the probability of packet loss due to media errors. Thus, XTP introduces *rate control* and *burst control*. When end-systems negotiate the quality of service of their connection, the receiver can throttle the transmitter by imposing a maximum rate (in bytes/sec) and a maximum burst (in bytes/burst) on the sender. Significantly, these parameters apply to the *path* followed by the data, and network routers are permitted to adjust these parameters dynamically in order to avoid congestion. Figure 1 shows how rate and burst control affect transmission.

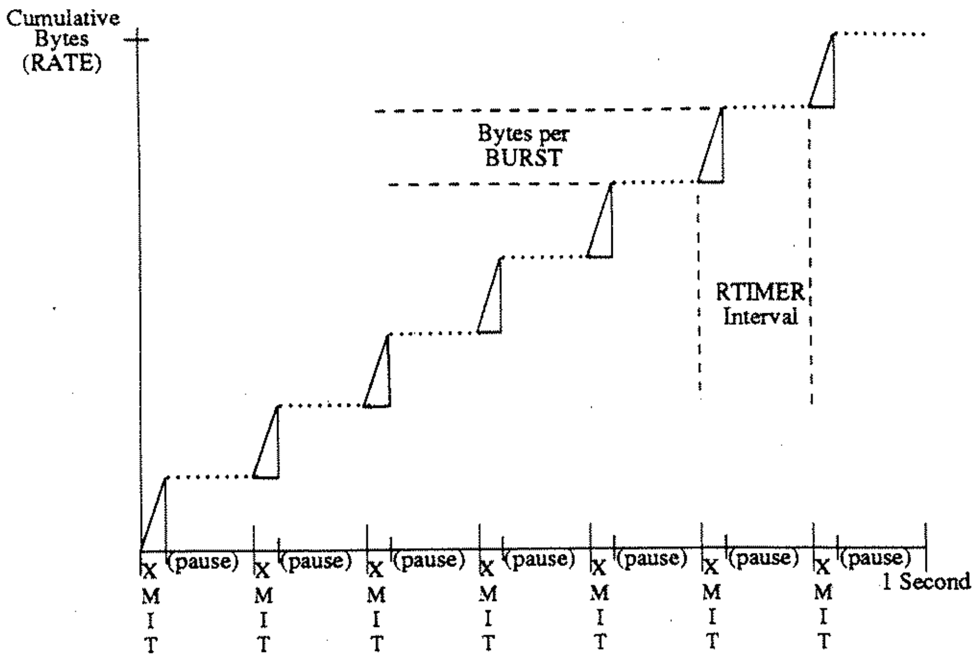

Figure 1.
Effect of Rate and Burst Control

Message Priorities. Most datalink protocols provide a priority mechanism that can be used to discriminate among user data types of varying importance (Ethernet is the notable exception; it provides no such capability). XTP allows this discrimination to be visible at the level of the user's message. Each message can be tagged with a 32-bit indication of its importance, and all packets which result from the segmentation of that message inherit that message's priority. At each scheduling opportunity, a node selects and operates on its most important packet. Thus it can be said that, with a granularity of one packet's transmission time, an XTP implementation is always working on its most important packet. Better yet, this operational definition is applied consistently, meaning that it is operative within all interior routers. This is clearly an essential feature to support real-time applications where latency control is paramount.

Out-of-Band Data. It is sometimes useful to send information *about* the data stream without embedding it *within* the data stream. As an option, XTP can carry with each packet up to 8 bytes of *tagged data.* Tagged data is passed from the transmitter to the receiver, and its presence is indicated to the receiver, but it is never interpreted by XTP. Tagged data is expected to be useful to upper layer protocols (e.g., presentation layer) for providing semantic information about the data. Tagged data also provides an expedient way to timestamp data being transmitted.

Selectable Error Control. XTP provides three types of error control. Traditional error control (provided by transport checksums) is provided for normal data and results in fully reliable, error-checked transmission. As an option, a receiver can generate a fast negative acknowledgement in order to speed the retransmission process. This will prove useful in environments (like local area networks) where out-of-sequence data implies lost (as opposed to delayed) data. Another option is *no-error mode* which suspends the normal retransmission scheme. Correctly received data is properly sequenced, but gaps are not retransmitted. This mode is expected to find utility when carrying digital voice and video.

Selectable Flow Control. As with error control, three orthogonal options are provided. Traditional flow control is available for normal data. *Reservation mode* practices a conservative flow control policy whereby the receiver may only issue a credit for buffers dedicated to this connection; this assures that data will not be lost due to buffer starvation at the destination. The third mechanism is to disable flow control entirely by using the *noflow* option. Note that such a "free flow" or "streaming" mode of operation is not available in other protocols; this mode may prove useful for multimedia applications.

Policy vs. Mechanism. XTP goes to great lengths to separate policy from mechanism. XTP assumes that the user knows best how to optimize the parameters of a connection. Said another way, the protocol designer knows the least about how any individual application will use the protocol, so the designer's role is to provide capabilities, not enforce policies. Examples of this separation are evident in the discussions of selectable flow control and selectable error control (above), and in the discussion of selective acknowledgement and selective retransmission in the next section.

Multicast. Traditional transport protocols make the assumption that a connection establishes bidirectional communication between two peers. But as control systems become distributed, the need for group communication becomes more acute. XTP multicast permits any one transmitter to send data reliably to an arbitrarily large group of receivers. In the context of multicast, "reliable" transmission means that every active member of the receiver group will receive the data, and that any data errors encountered are transparently recovered by the protocol. However, multicast group management is intentionally outside the XTP definition, so operations such as dynamically joining, leaving, and rejoining an on-going multicast communication are handled separately (see [12] for a discussion of useful group management techniques). Multicast is ideal for updating multiple clients simultaneously, and so has obvious utility for applications such as sensor data distribution or maintaining multiple consistent copies of a database. Reliable transport multicast is arguably the most innovative single new feature in XTP.

3. PERFORMANCE ADVANTAGES

In addition to increased functionality, XTP offers the promise of higher performance, especially when implemented in hardware. Performance is enhanced by utilizing a header/trailer protocol design, fixing the position of option bits in the header, optimizing connection setup and address translation, integrating the transport and network layers into a *transfer* layer architecture, utilizing selective retransmission and selective acknowledgement, aligning data on long word boundaries, and, finally, promoting a VLSI implementation.

Header/trailer Protocol. Conventional protocols are header protocols, meaning that all protocol control and error checking information is in the message header. This implies that the transport checksum must be calculated and inserted in the header before transmission can begin, which in turn implies that the message must be complete before transmission can be initiated. XTP puts the 4-byte checksum in a protocol trailer, thereby allowing it to be computed and appended on-the-fly by appropriate hardware. Moving the checksum to the trailer permits "streaming" of data through the protocol, and also eliminates one data copy operation.

Fixed-position Control Information. All protocols allow the user to make selections from a set of options; however, this often results in variable length control information in the packet header. With an eye toward a hardware implementation, XTP defines option bits in the header, but fixes the position of the option bits regardless of whether the option is selected. This little bit of intelligent design clearly increases performance and decreases complexity in the silicon implementation.

Connection Management. Reliable data transmission in TP4 requires a three-way (six packet) handshake: two to request and confirm the connection, two to send and acknowledge data, and two to release the connection and confirm disconnection. Figure 2 below shows how an association can open a connection, send data, then acknowledge the data and release the connection with a three-packet exchange. As an additional optimization, XTP allows data to be carried in the first packet transmitted. While non-traditional, this approach anticipates a successful connection setup and allows data to be processed immediately thereafter.

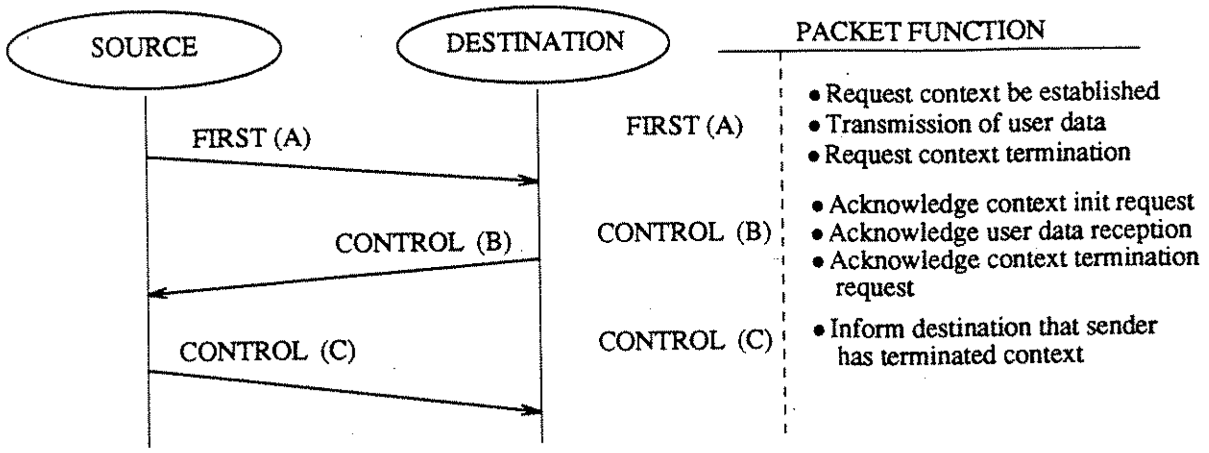
SOURCE
DESTINATION
PACKET FUNCTION
FIRST (A)
CONTROL (B)
CONTROL (C)
FIRST (A)
• Request context be established
• Transmission of user data
• Request context termination
CONTROL (B)
• Acknowledge context init request
• Acknowledge user data reception
• Acknowledge context termination request
CONTROL (C)
• Inform destination that sender has terminated context

Figure 2.
Connection Management

Address Translation. Traditional network layer protocols send a full network address with every packet, thereby reducing overall network efficiency. Common remedies for this include cacheing schemes or header prediction algorithms. The former becomes expensive when the number of network addresses simultaneously in use is large; the latter fails to increase performance when network traffic is random (e.g., X-window server). In contrast, XTP uses a key-based scheme in which the network address, carried in full in the first packet that establishes the connection, is resolved into a 32-bit *key*. After the first packet has been processed, all subsequent packets carry only the key, which is used as a simple index into a lookup table.

Network Addressing Plan. XTP does *not* introduce (yet another) network addressing plan. The first packet on a connection contains a parameterized address consisting of an *address type* and an *address segment.* Network addresses are resolved by reference to the addressing plan for the selected address type. Currently, 12 different addressing schemes are identified, including Internet and ISO 8348 network addresses. Other popular types could be added on demand.

Transfer Layer Architecture. XTP is a *transfer* layer architecture, patterned after the French military standard for real-time local area networks called GAM-T-103 [6]. The transfer architecture recognizes the efficiency of providing end-to-end reliability and network routing in the same layer, thus enabling the concept of an end-to-end "silicon path." This collapse of the traditional transport and network layers into a single transfer layer is easily the most contentious design decision in XTP, and probably will not survive the standardization bodies. Nevertheless, as currently specified [10], XTP achieves a speedup from this integration.

Selective Retransmission. TCP and TP4 use *go-back-n* retransmission; i.e., when an error is detected, the transmission window is reset and begins again with the first byte (in TCP) or packet (in TP4) which was lost or received in error. Thus, depending upon circumstances, *go-back-n* may retransmit data which has already been received correctly. XTP can use either *go-back-n* or *selective retransmission*; in the latter, the receiver acknowledges spans of correctly received data and the sender retransmits only the gaps. The user selects which scheme to use, if any (recall that retransmission can be defeated entirely using no-error mode).

Selective Acknowledgement. Acknowledging every packet (1) assumes that the network often loses packets, (2) assumes that the transmitter wants acknowledgements, and (3) embeds policy with mechanism. XTP allows the user to decide if and when acknowledgements are desirable. Acknowledgements are provided whenever the transmitter requests them, thereby allowing the user to select an acknowledgement frequency ranging from always to sometimes to never. Philosophically, acknowledgement generation from the receiver is decoupled from data arrivals or window sizes. Whenever the transmitter wants to know the status of the connection, it asks; when the receiver responds, it tells everything it knows about its current status.

Data Alignment. Although it is a simple notion, data alignment pays big dividends. First, the major fields of the header/trailer are aligned on 4-byte boundaries; this minimizes the number of memory accesses needed to retrieve a field. Second, the user's data itself need neither begin nor end on a 4-byte boundary, although the transmitted frame will do so. An *offset* field identifies the start of the data in the information field, while a *length* field identifies its end. This technique makes it simple to handle the case when the user data is smaller than the smallest allowable frame.

4. WHY THIS PROTOCOL?

In summary, XTP provides six major advantages:

(1) It accommodates the realities of modern networks: high data rates, densely packed bit-pipes, low bit error rates, and different data transfer paradigms supported simultaneously by a single protocol.

(2) The protocol is programmable in the sense that XTP provides mechanisms while the user sets policy. This reflects the observation that the user is best equipped to judge what data transfer paradigm is most appropriate for a given application.

(3) XTP is evolutionary. It preserves the common features found in extant protocols, while simultaneously extending them in support of distributed applications.

(4) The transfer layer architecture integrates the operation of the transport and network layers. End-systems can be more tightly coordinated because intermediate systems (routers) participate in rate and burst control.

(5) XTP supports parametric addressing. Rather than introducing yet another network addressing plan, it can potentially work with any scheme. It specifically recognizes the dominance of Internet addressing and ISO 8348 addressing.

(6) XTP retains a basic finite state machine design which encourages implementation in silicon.

5. THE PROTOCOL ENGINE

The *Protocol Engine* (PE) is a multi-chip, programmable device which seeks to off-load from the host many of the tasks associated with the network subsystem. To do this at media speeds requires a "flow-through" architecture such that packet processing can be pipelined. Figure 3 shows the processing timeline when the PE is connected to an FDDI network in which the shortest legal packet is 60 bytes in length. Thus, at the 100 Mbits/sec data rate of FDDI, this permits only 5 microseconds for packet processing.

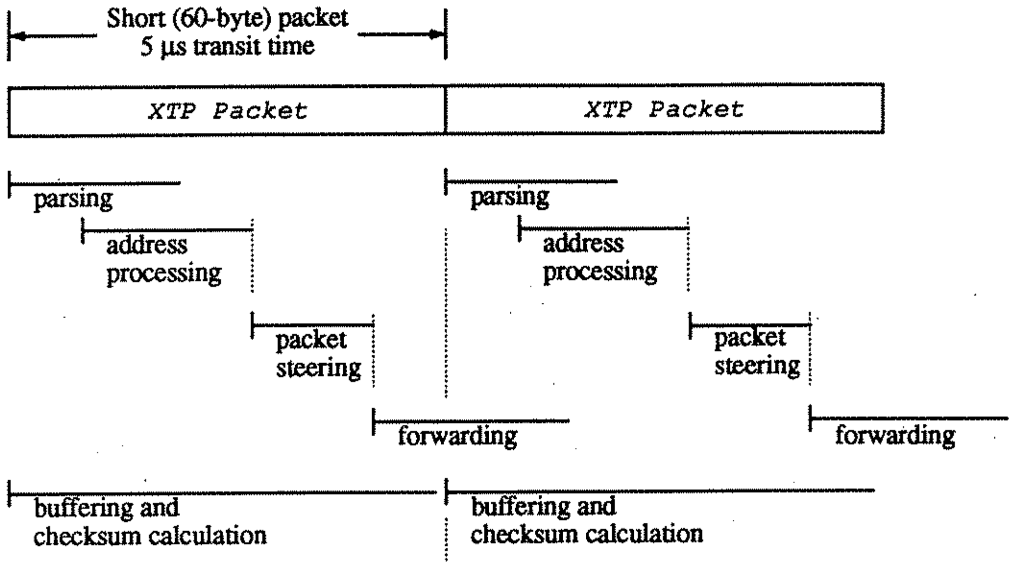

Figure 3.
Protocol Engine Packet Processing Timeline

To achieve such a high degree of performance, processing functionality is divided into four distinct VLSI chips as shown in Figure 4. Each chip is composed of a microprogrammable core processor, sequencers, FIFOs, and bus interface logic. Two busses, the data bus (DBus) and the control bus (CBus), allow the chips to communicate without disturbing the primary stream of network data.

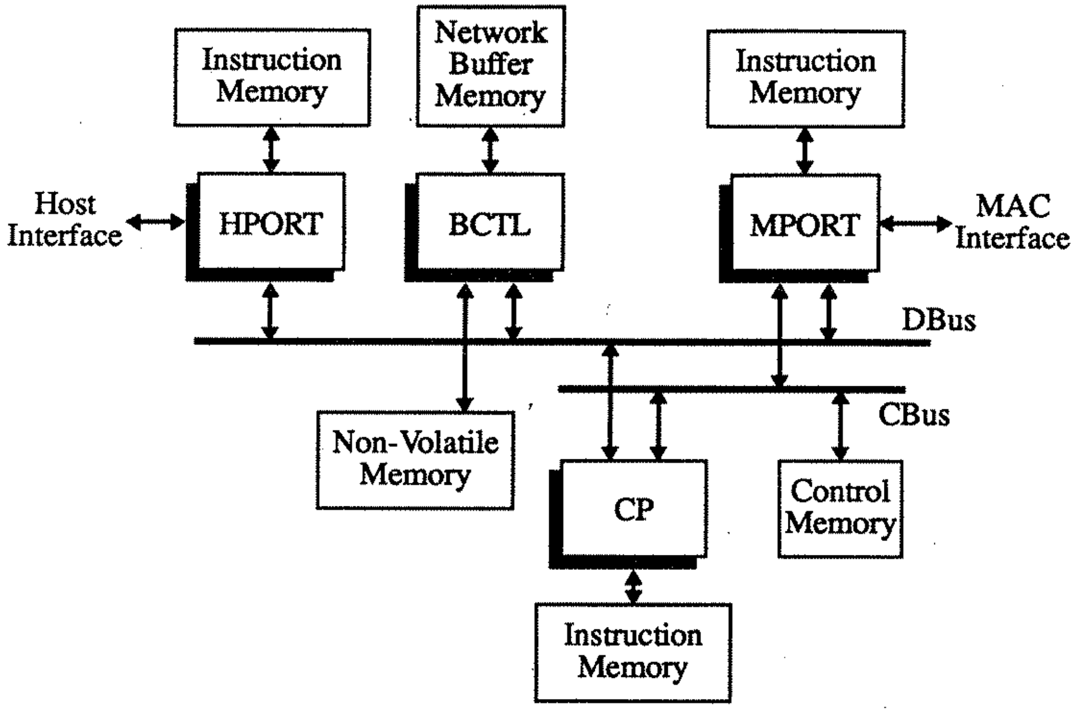

Figure 4.
Protocol Engine Block Diagram

MPORT. The MAC Port (MPORT) connects the PE to a particular network's Medium Access Control (MAC) hardware. The MPORT moves data bidirectionally between the MAC device and the high-speed network buffer memory. On its receive side, the PE's MAC device will filter incoming packets based on their destination MAC addresses; if destined for this device, they are then passed to the MPORT. The MPORT validates the checksum in hardware and extracts protocol control information from the header; the packet payload is demultiplexed according to its context (i.e., the connection to which it applies). Protocol control information is passed on the CBus while the packet itself is passed on the DBus. On its transmit side, data is moved from the network buffer memory, across the DBus, into a transmit FIFO (which corrects timing problems between the speed of the DBus and the speed of the network), and finally to the MAC device for transmission.

HPORT. The Host Port (HPORT) processes data flowing between the network buffer memory in the PE and the memory in the host, including the interface to the host's backplane bus. When the host has data to transmit, the HPORT interrogates the context, generates the packet header, calculates the checksum and generates the trailer, updates the sequence number information, and performs the utility functions of byte swapping (if needed to go from a little-endian to a big-endian machine or vice versa), word alignment, and padding. When the HPORT is receiving data from the network, it separates protocol information from payload data, delivering them separately to the user memory in the host.

BCTL. The Buffer Controller (BCTL) is the simplest of the four chips. It provides the interface between the DBus and two memory subsystems, the network buffer memory and non-volatile memory. The BCTL allows the DBus to have direct access to the network buffer memory when data is flowing between the MPORT and HPORT. Like the MPORT and HPORT, BCTL is programmable, and its non-volatile memory is used for instruction storage.

CP. The Control Processor (CP) implements high-level protocol processing functions such as association management and path management. The CP can be a microprogrammable RISC processor, or its functionality can be accomplished by the host. Whereas the MPORT, HPORT, and BCTL perform generic functions, the operations of the CP are protocol-specific. The first three chips together form the *Protocol Accelerator*, while the four chips together make a *Protocol Engine.*

The Protocol Engine is currently being developed commercially by Protocol Engines Inc., Santa Barbara, California. The protocol accelerator chips are expected to be available in late 1992. A board-level product interfacing the protocol accelerator chips to a commercial FDDI MAC on the network side and to an SBus or VMEbus controller on the host side is expected in early 1993.

5. SUMMARY

In summary, the Xpress Transfer Protocol offers a variety of new functionalities which are thought to be needed in the distributed systems to be built in the 1990s. XTP's realization in the Protocol Engine represents a very high performance network processing subsystem, and one which addresses the real-time messaging requirements of factory automation.

6. 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, and the Virginia Center for Innovative Technology. Within the Computer Networks Laboratory at the University of Virginia, we especially acknowledge the XTP implementation work of Robert Simoncic and John Fenton, and the XTP analyses and descriptive text prepared by Tim Strayer and Bert Dempsey.

7. REFERENCES

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

2 J. Postel, ed., "Transmission Control Protocol - DARPA Internet Program Protocol Specification," RFC 793, USC/Information Sciences Institute, September 1982.

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

4 R.W. Watson, "Delta-t Protocol Specification," Lawrence Livermore Laboratory, April 15, 1983.

5 D.D. Clark, M.L. Lambert, and L. Zhang, "NETBLT: A High Throughput Transport Protocol," Network Information Center RFC 998, SRI International, March 1987.

6 French Ministry of Defense, "GAM-T-103 Military Real-Time Local Area Network Reference Model (Transfer Layer)," February 7, 1987.

7 D.R. Cheriton, "VMTP: Versatile Message Transaction Protocol, Protocol Specification," Network Information Center RFC 1045, SRI International, February 1988.

8 A.G. Fraser, "The Universal Receiver Protocol," *Proceedings of the IFIP Workshop on Protocols for High Speed Networks*, Zurich, May 1989.

9 G. Chesson, "Datakit Software Architecture," *Proceedings of the ICC*, pp. 20.2.1-20.2.5, 1979.

10 "The Xpress Transfer Protocol," Protocol Engines Inc., Santa Barbara, California, January 1992.

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

12 B.J. Dempsey, "An Analysis of Multicast and Multicast Group Management," M.S. thesis, Department of Computer Science, University of Virginia, Charlottesville, Virginia, January 1991.