-- --

The Evolution of XTP†

Greg Chesson

Silicon Graphics Computer Systems
Mountain View, Ca 94039-7311
greg@sgi.com

Abstract

The design of XTP has been a 3 year process so far with roots traceable to work in the 1970's and 80's. There have been roughly ten distinct revisions of the protocol. Since Revision 3.4 in June 1989 there has been an effort to stabilize the design. This has been accomplished. The most recent versions - Revision 3.4 and 3.5 - were about 14 months apart whereas earlier versions were 3 to 6 months apart. Fortunately the net effect of the change from Revision 3.4 to 3.5 was to remove mechanism from the design.

There have been at least nine distinct software implementations of the protocol. Experience with these implementations has been fed back into the protocol specification. We believe that the functionality in the present design, Revision 3.5 plus Addendum 1a, is strengthened by the number of alternative mechanisms that have been tried and rejected.

XTP history and evolution will be reviewed in this paper with a focus on some of the protocol mechanisms that have changed over time.

Introduction

The quantity of network and protocol research can be measured by observing the number of publications in the area. In the 1960's and early 1970's most of the papers published in the biennial ACM SIGOPS Proceedings were devoted to operating system topics like paging, process control, and file management. By the 1980's the balance changed so that most of the papers presented some aspect of networking in the context of operating systems. The papers presented in the first workshop in a series dedicated to protocol research [9] reference much of the work of the 1980's and suggest that the 1990's may be a time of significant progress in this area.

Contemporary protocol research is influenced by developments in other technical fields. For example VLSI techniques are commonly incorporated into the design of protocols, systems, and media - something that was not possible when many standard protocols were designed. Improved media in the 100Mbit/sec to 2000Mbit/sec range limit the usefulness of protocols designed for slower media. As a result updated methods for flow, rate, and error control and new services receive attention. Applications such as advanced CAD and multimedia demand greater bandwidth and low latency interfaces. As a result protocol mechanisms for providing bulk and media services, transactions, multicast, and network resource allocation become important.

† published in the Proceedings of the Third International Conference on High Speed Networking, North-Holland, 1991.

Protocol Evolution

A number of writers offer insight on the current state of the art. The final report of the ANSI X3S3.3 Study Group on High Speed Network Protocols [7] documents changing conditions and requirements that justify updating existing protocols or designing new ones. In a paper on emerging protocols Ian Wakeman[8] examines several technical areas in which current protocols can be considered deficient. The study published by the IBM Research Laboratory in Zurich [6] compares a number of protocol designs with each other and with the requirements for very high speed networking.

The Xpress Transfer Protocol (XTP) is a new protocol design that has been carried out by a multidisciplinary group including a VLSI design team, an operating system team, protocol architects, and real-time system experts. There has also been feedback from several software implementations. The evolution of XTP is defined by the interaction between these various interest groups over a period of years.

XTP History

The chronology of XTP begins with work done at Bell Telephone Laboratories in the late 70's and early 80's by the author and others on lightweight protocols† for the Datakit® network - a research project that can be described as the forerunner of today's ATM architecture. An overview of this work with references to other developments may be found in the bibliographic notes of Holzmann's book on protocol design[11]. This book also describes the *Promela* language and *Spin* software that have been used to formally model parts of XTP.‡

During this same period several researchers at Bell Laboratories were practicing the art of rendering software algorithms into special-purpose hardware. First in this area was the chess machine, *Belle*, designed and built by Ken Thompson and Joe Condon. Similar methods were pioneered at Silicon Graphics in the early 1980's with the development of the Geometry Engine® for accelerating graphics systems.

A common thread to these projects was that the developers did not simply apply VLSI techniques to existing algorithms: they had to transform their system algorithms to take advantage of and fit the constraints of hardware acceleration. The resulting systems are a hybrid of software, firmware, and special hardware. This suggests that VLSI techniques can be applied to any protocol, but will work best if the protocol can be adapted to the new environment. This is why hardware considerations have always played a part in XTP architecture.

Work leading to XTP began in late 1986 with a presentation to the ANSI X3T9.5 group that was sponsored by Larry Green who was then co-chair of the SMT committee. The presentation was entitled "RISC Protocols". This confused a lot people who thought the idea was to use a RISC cpu to implement protocols. The real idea was to apply the principles of RISC cpu design to protocol and protocol-VLSI design:

minimalist functionality == high clock rate.

The initial meeting with ANSI led to publication of several white papers[3] and the adoption of a new name, *Protocol Engine*®. We think of these papers as XTP Revision 1 which was basically a trailer protocol. XTP Revision 2.0, produced in September 1987,

† The term *lightweight* was first used by the members of the Computer Laboratory at Cambridge University who developed the Cambridge Ring and its Byte Stream protocol.

‡ The *Spin* software is available at no charge by sending electronic mail to *netlib@research.att.com*. The mail daemon there will respond with detailed instructions.

was an improved design that looked more like a protocol specification. Reaction to these designs fell into three categories: enthusiastic, we'll wait-and-see, and we'll stick-with-TCP (and so should you). We were obviously mandated to proceed.

Late in 1987 Larry Green engineered the concept of a Technical Advisory Board (TAB) of member companies and research groups that could support the continued development of XTP with funding, engineering reviews, testing, and general advice. His work led to the creation of a TAB for XTP development.† The first meeting was held in Santa Barbara in February 1988. Although the TAB membership has changed during three years, it is active and strong and largely responsible for XTP achieving its present level of development.

Revision 3.0 of XTP was distributed at the first TAB meeting. There was also a demonstration of the first XTP implementation - a user space implementation using raw ethernet interfaces to communicate. Revision 3.1 of the protocol specification was produced March 28, 1988. This was followed by Revisions 3.2, 3.2.5, 3.3, 3.4 in June 1989, 3.4.5 (a TAB editor's document), and 3.5 in September 1990.

An addition to Revision 3.5 was decided upon by the TAB in December 1990 and was produced in the form of an Addendum. Addendum 1a fixes some protocol deficiencies that were discovered in 3.5. It removes some mechanism from the protocol, e.g. SUPER packets, and clarifies several sections of text. Although we disliked publishing the Addendum because it creates a second document, the next document revision (3.6) is expected to go through a more lengthy review and editing process than earlier revisions. It seemed more important to deliver current design information to those studying XTP, particularly researchers who may not have easy access to the protocol design team.

Design Goals

We wanted a protocol that would facilitate the design of supporting VLSI and which would be suitable for operation at gigabit speeds. A simple design based on fixed-size headers and trailers and no variable-size options was indicated. On the other hand we didn't want a simplistic protocol that might not be able to provide the services needed by modern distributed systems.

It was decided that several service models - message, stream, transaction, bulk, controlled rate, and multicast - were needed. It was observed that with the exception of multicast, these services are all the same: deliver some bits between point A and point B. The difference between these services is that they represent different distributions of data traffic and frequency. Consequently we could try to design a protocol that could supply these different services, including multicast, with a single mechanism.

Of all the remaining goals and features in XTP the most important may be the way that XTP relates to media, switches, and Layer 3 services. A super efficient transport

† The first TAB company was Interphase, followed by Apollo Computers. The corporate TAB membership currently includes AMD, Artel, Boeing, Concurrent, Crosfield Electronics, DY-4, E-Systems, IBM, Intel, Intergraph, Interphase, Linotype-Hell AG, Lockheed, SBE, Scitex, Unisys, and Xerox. Non-corporate members, called research affiliates, include the Admiralty Research Establishment, Bowman Gray School of Medicine, Concordia University, Electronics and Telecommunications Research Institute (of Korea), Laboratoire de Genie Informatique, Johns Hopkins University Applied Physics Laboratory, NASA, National Center for Supercomputer Applications, Naval Ocean Systems Center (NOSC), Naval Surface Warfare Center (NSWC), Superconducting Supercollider Laboratory, University of Kentucky, University of Virginia, and University of Melbourne.

protocol will not be very useful if it must operate over Layer 3 services that deliver far less performance. Also, trends in media design are causing a gradual obsolescence of traditional Layer 3 architecture. The reason is that media providing switched virtual circuits such as ATM change the way we think about Layer 3.

This suggests that a modern protocol should anticipate an environment populated by both switching and non-switching media. The protocol should operate well with either kind of network and also provide a means for coupling between dissimilar media.

The XTP approach is to provide what some would call a Layer 3 routing service, but what is in reality an emulation of an ATM switch based on the "cut-through" model of switching. The XTP header remains the same whether or not the facility is used. The ROUTE field in XTP may be used to select a path through an XTP switch/router or as a virtual circuit identifier for a hardware switch. Otherwise XTP can be used with a standard Layer 3 such as the Internet IP protocol which designates protocol type 36 (decimal) for XTP.

Early Ideas

An early concept in XTP design was, and still is, the notion of "real-time" or "flow-through" processing. The idea is to pass data units, or packets, through a pipeline of processing engines. The bidirectional pipeline would connect to a network at one end and to a host computer system, router, or other device at the other end. The data rate through the pipeline should equal or exceed the media data rate. If protocol and system i/o processing can be completed within the pipeline, at the media data rate, then the network and i/o processing can be termed "real-time".

With real-time processing the bandwidth of the media can be directed in or out of the host system with little queuing delay or bandwidth loss. Second, the latency for sending or receiving messages can be quite low if the pipeline depth is shallow. This is important because techniques for increasing bandwidth usually increase latency, and techniques for reducing latency usually decrease bandwidth. With a small number sufficiently powerful processing engines the pipeline approach can provide both high bandwidth and low latency.

XTP designs came and went as designs for a hardware pipeline came and went. But the goal of achieving real-time packet processing has never changed. Note that the transit time of a 60 byte packet, which is the size of a TCP ACK, on an FDDI network is just 4.8 microseconds. At 1 Gbit/sec it becomes .48 microseconds.

Measurements show that 60% to 80% of the traffic on a current IP/TCP network are between 60 and 200 bytes in length. This suggests that real-time processing can be an important tool for reducing congestion at hosts, routers and switches.

One of the earliest ideas for XTP was to create a tight binding between application buffer locations and a data descriptor, usually a sequence number, in the protocol. In one design data would be sent through the network in predetermined, fixed-size packets. In most cases the sizes were powers of two. Each packet would be represented by a bit in a bit vector. The familiar operations of representing a window in a numeric sequence space and modular arithmetic could be represented in terms of boolean operations on bit vectors. For example a flow control window was represented by a bit mask. Testing to see whether a packet was inside the window was represented by an OR operation.

The motivation for the boolean scheme was to define a protocol whose algorithm could be implemented with simple logic gates. This amusing design was presented at a workshop in 1987. A quick analysis of the bandwidth-delay product on a gigabit link shows that the bit vectors needed to support a high speed stream would be immense. Bit vectors were quickly replaced by some designs involving structured sequence numbers.

All of these design tricks created problems with retransmission, fragmentation, and other parts of the protocol. The method that works best when packets may have to change size during processing is to sequence number each byte. One must choose sequence numbers with sufficient precision to enumerate a high-speed stream.

In the end 32-bit sequence number calculations can be done at sufficient speed that the bit vectors and other techniques weren't needed - the hard problems were elsewhere, starting with address translation.

Address Translation

Address translation, whereby address bytes from a header are used as the lookup key to a data base lookup procedure, can be the most time consuming component of input protocol processing. It was thought that a lookup procedure that would operate in a fixed amount of time would be best for a hardware pipeline. It was realized that a hardware implementation of a trie-based lookup scheme would be best for this purpose. Content-addressible memories were ruled out for size and cost reasons.

Early XTP proposals incorporated three 32-bit destination address components in each header: destination network address, host address, and port address. These were followed by a three component source address. Since a trie lookup engine could be designed to perform address translation in fixed time, it might as well be used on every packet. There are a few problems with this assumption. First, as pointed out by A. G. Fraser, the overhead of large addresses would be unsatisfactory if the protocol were used on slow media - addresses would use an unacceptable fraction of available bandwidth. Second the trie memory space can become enormous in applications storing thousands of addresses. Finally even though a hardware-based implementation might be capable of performing an address lookup on every packet, a software implementation would view this as a constant and possibly expensive processing overhead.

Address lookups can be accelerated using cacheing schemes. Although these are easier to implement in software than hardware, techniques such as Van Jacobson's header prediction scheme for TCP can be helpful. Unfortunately a header prediction technique does not reduce software processing overhead when the prediction fails. Applications like X-Windows cause a server to receive random short messages from different systems. If the header cache size increased to accommodate the most active connections, then the cache lookup problem has about the same overhead as the original address translation problem.

For these reasons XTP evolved to the KEY-based scheme employed today where there are three items extracted from a packet for address translation: MAC source address, 32-bit KEY field, and 32-bit ROUTE field. Once an initial handshake has been completed between end systems, address translation reverts to a single index-based table lookup. This has about the same overhead as a header prediction scheme, but works on every packet.

The initial packet that sets up an XTP connection, called a FIRST packet, contains a variable-length *address segment*. The syntax of an address is specified by a *type* field. XTP specifies several address formats for compatibility with the Internet world, the ISO universe, and other addressing plans. This notion of a *parameterized address* may be unique to XTP - it enables the addition of XTP to existing networks without introducting an additional address plan.

The design of Protocol Engine hardware pipelines progressed from trie-based schemes to hash-based schemes. Hardware was designed to extract address bytes from a passing packet according to protocol type, and generate a CRC-based hash function. A specialized processor is used to complete the hash lookup. The processor has special instructions for reducing the time required for comparing binary strings, and a fifo ensures that the occasional long hash chain does not cause input processing to drop packets.

Note that the hash lookup is needed only for FIRST packets or until a so-called KEY exchange has been completed. Non-FIRST packets are processed with a table lookup. The result is that hashing is needed for only a small percentage of packets.

Even though an indexed table lookup is probably the simplest form that address translation could take, there is still overhead involved in mapping a potentially long parameterized address segment to a KEY value during the processing of a FIRST packet. Real-time systems often maintain a fixed assignment of protocol connections, called *Permanent Virtual Circuits* (PVC). We wanted XTP to have a PVC facility for use in real-time systems.

Until Revision 3.5 the capability was implemented by a DADDR bit in the control word of the header. If this bit was set it meant that the KEY field should be interpreted as a permanent virtual circuit id. It also meant that an address segment would not be present in a FIRST packet and that ordinary address processing for a FIRST packet would be eliminated. This seemed like a reasonable approach but added unnecessary complexity to normal packet processing. The DADDR bit had to be tested on every packet and had the effect of adding extra logic to FIRST packet processing. Consequently, DADDR was decommissioned as a header bit and replaced by an address type, also called DADDR. FIRST and address segment processing now have no special cases, and the address processing for DADDR consists of using the KEY as a PVC as before.

The DADDR experience illustrates how option processing in a protocol design can impede the normal case when the option is not used. Unfortunately multicast addressing processing does not merge into the normal flow as neatly as DADDR processing. Consequently the MULTI bit in the header, which indicates multicast, is tested at the beginning of address processing. If the bit is set the rest of the logic is replicated to contain the number of additional tests that would be needed in the non-multicast case.

Framing

XTP was first a trailer protocol. That is, it did not have headers. This configuration didn't last long however, because the hardware pipeline needs to know the destination address, packet type, and sequence number when a packet arrives so that data can be "steered" to the proper location and the sequence number be checked. Any other control information needed before data delivery should be in the header. Conversely, checksums and some control bits that take effect after data delivery are best located at the trailing end of a packet. Checksums can be accumulated and then checked or inserted at the end of the frame. Connection shutdown indications or an end-of-message (EOM) condition would normally be acted upon after data delivery. If such control bits were in the header, they would have to saved in one of the pipeline stages for later use, so it is more useful for such information to follow data through a pipeline.

All versions of XTP have incorporated fixed-size headers and trailers aligned on multiples of eight bytes. Eight byte alignment is provided for programming efficiency when the protocol is implemented in software on a 64-bit machine. It also anticipates future Protocol Engine designs that might have internal 64-bit data paths.

The headers and trailers do not change size when options are selected. Rather, the bitfields for all protocol mechanisms are present at all times.

Until recently all versions of XTP located the trailer by counting backwards a fixed number of bytes from the end of the packet. This can work but it has two problems. First, the hardware needed to do this in a VLSI pipeline is trickier than originally believed. Second, bridges and other devices may not always preserve the length of a packet. So any additional padding bytes added to a packet make it impossible to find the trailer by counting backwords. For this reason a reserved field in the XTP header was defined as a *length* field. This made packet processing "safe", and eliminated reverse parsing logic from hardware pipelines. The length field and related simplifications are explained in Addendum 1a to Revision 3.5 of XTP.

SORT Field

XTP has incorporated a priority mechanism for both input and output ordering. Prior to Revision 3.5 the field could be interpreted in two ways: either as a 32-bit static priority, or as a 32-bit time deadline. More hours of discussion went into SORT field semantics and related real-time issues than almost any other aspect of XTP design.

Unfortunately no plausible method for converting between the two representations, or specifying how the two mechanisms could interperate, was discovered by those interested in this aspect of XTP. For these reasons the SORT field definition was simplified to being just static priority. The issue of representing more advanced scheduling metrics in the protocol remains a research issue. It is hoped that we will be able to return to this issue after gaining experience with XTP in advanced real-time systems.

Record Structure

Protocols are often used to transmit sequences of records or messages whose binary image is then reconstructed at a receiver. Protocols are also used to provide a byte stream service. What we wanted in XTP was a byte stream service that would also provide message facilities adequate for record marking, encapsulation, and convergence protocols.

We define record marking as the facility to send a sequence of arbitrary byte length messages on a byte stream protocol and to reconstruct the original messages at the receiving end. We define encapsulation as the ability to transport protocol data units produced by a protocol such as IP/TCP or ISO TP4 as data units in XTP. The transported data units would be carried through an XTP subnetwork as data and injected into a native IP/TCP network or ISO TP4 network at an XTP gateway. We define a convergence protocol as one that provides encapsulation as just described, but also provides the service of the encapsulated protocol. An example of convergence is provided by a modified NFS implementation, which normally uses UDP, but instead multiplexes NFS transactions to a server over a shared XTP association. The method for representing the NFS transactions and delivering them to the NFS software is an example of a convergence protocol.

Message service can be provided with a stream protocol by two basic methods: either impose a record structure on the stream, or use a record delineating mechanism provided by the underlying transport. The imposition of a record structure on the stream is easy, but potentially costly since it requires scanning and parsing the data stream. Reliable stream protocols seldom expose the underlying frame structure to the stream user. One notable exception is the Sequenced Packet Protocol (SPP) defined as part of the XNS protocol suite. This protocol provides an 8-bit Data Stream Type that is settable on output and delivered on input.

XTP provides a method for marking the data stream. The EOM bit in the trailer, borrowed from the XNS mechanism of the same name, accomplishes this. For encapsulation the protocol provides for a 64-bit *tag* field at the beginning of the application data segment. The presence of the field is indicated by the BTAG bit in the header.

Until recently XTP provided tag fields at both the beginning and end of a data segment using BTAG and another bit, ETAG, to indicate the presence of these fields. BTAG appeared in the header, and ETAG in the trailer. Because ETAG was redundant and because the *etag* field could also be non-aligned on a modulo-8 boundary, it was removed from the protocol.

Checksums

The XTP checksum was always intended for hardware implementation in a parallel data path. This meant that checksum logic could easily become part of a host DMA engine. It also meant that a software implementation could do 32-bit or 64-bit fetches from memory instead of fetching one or two bytes per algorithm step.

The mistake made with early versions of the XTP checksum was that because it was parallel, it was not portable. That is, a little-endian cpu and a big-endian cpu could compute the checksum quickly, but would get different results. The choices for correcting the problem were to force one of the cpu types to reorder bytes before checksumming, or to change the algorithm. The algorithm was therefore changed to its present form, called CXOR.

Sequence Numbers

Most quantities in XTP control structures are 32-bit values. In older versions a provision was made for also having 64-bit values.

The fixed-size structures of XTP allow for a stateless conversion between 32-bit and 64-bit implementations. It was shown that 32-bit implementations could interoperate with 64-bit versions through a translating gateway or by a simple conversion process. The trick is that the fixed size headers and alignment rules allow a simple doubling or halving of header and trailer sizes. An additional bit is needed in the header to direct a 64-bit implementation to truncate its arithmetic to 32-bits.

This is one example of a mechanism that was removed from the protocol, but which could be revived when multi-gigabit networks are deployed.

Error Control

Experience with Datakit where the hardware provides a switched order-preserving packet stream suggested that a go-back-N retransmission model would be sufficient for high speed applications. This simple model is not best for all applications. The pathological cases occur over slow media, in environments that can reorder packets, and in high bandwidth-delay environments. In these cases a selective retransmission scheme is useful whereby only error frames are retransmitted.

The first definitions of XTP used only go-back-N. When a selective retransmission method was added, the model incorporated the notion of sequence number *spans*. Each span is an ordered pair of sequence numbers. Spans originally designated missing data, or sequence *gaps*, detected by a receiver. Later the spans were shifted so that instead of representing gaps of missing data, they represented the islands of correctly received data.

Care was taken throughout to guarantee that an implementation could provide go-back-N service and still interoperate with another implementation that provided selective retransmission based on spans. Since both the algorithm and the number of spans can be chosen by the implementor, this part of the design has been called *selectable retransmission*.

Multicast

The multicast facility in XTP has gone through several transformations. Early versions provided both multicast and concentration. All versions of multicast and/or concentration provided error control as an option. Early versions of concentration used the trick of acknowledging several messages in one multicast transmission. However, it was determined that applications of greatest interest to the TAB group involved the simple single-sender/multiple-reader model and that concentration when required could be provided by unicast (point-to-point) connections. So the multicast part of the protocol was simplified.

Between Revisions 3.4 and 3.5 implementation groups began to experiment with multicast in an effort to characterize and improve the performance of a multicast stream. This work led to an implementation technique for flow control called the *bucket algorithm*. This procedure and other multicast techniques are documented in Revision 3.5 and are not discussed here. What has not been documented before is the design choices made in non-multicast that enable the multicast mode.

First there is a MULTI bit in the header that is set on multicast output and tested on input. This bit simplifies special case testing where multicast processing is different from unicast. More important are the *sync* field and the SREQ bit. The SREQ bit requests an immediate state acknowledgement from the receiver. Not all protocols have this mechanism. XTP uses it for many things: determining round trip times, determining when to retransmit, and for generating a flow of state acknowledgements that are controlled by the protocol sender. The ability to control acknowledgements lets XTP divide the output flow into time slices called *buckets*. The *sync* field is used to identify each bucket and to identify any acknowledgements that were caused by an SREQ sent during a bucket interval.

This kind of information is overkill for managing the flow of a single stream, but in XTP multicast the flow control processing must be spread out over time, i.e. some number of buckets, to get adequate error control. The ability to manage the flow of control messages and identify output epochs are critical components of the multicast procedure. By designing them into the unicast procedure we avoid having to add redundant functions to the base protocol for supporting multicast.

Acknowledgements

The architecture has been influenced by many other designs including Delta-T, VMTP, Datakit protocols, Netblt, Blast protocols, ATM, IP/TCP, GAM-T-103 [1], the Cambridge Ring, and XNS. More people have contributed to the evolution of XTP than can be named here. The greatest concentration of effort has been through the TAB group and its Research Affiliates, the ANSI X3S3.3 High Speed Protocols Study Group, and the design and implementation teams at Silicon Graphics Computer Systems and Protocol Engines Incorporated. The support and encouragement of SGI and PEI to pursue this

work is gratefully acknowledged.

REFERENCES

[1] Ministre de la Defense, Republique Francaise, *"Military Real Time Local Area Network"*, GAM-T-103, February 9, 1987

[2] Protocol Engine and XTP are registered trademarks of Protocol Engines, Incorporated.

[3] G. Chesson, *"Protocol Engine Design"*, in Usenix Conference Proceedings June 1987, p. 209, also *"The Protocol Engine Project"* in Unix Review, September, 1987, also *P-Engine Protocol Definition*, September 1987, Protocol Engines Inc., 1900 State Street, Santa Barbara, Ca 93101.

[4] *"XTP Definition, Revision 3.5 plus Addendum 1a"*, September 10, 1990, Protocol Engines Inc., 1900 State Street, Santa Barbara, Ca 93101.

[5] William D. Schwaderer, *"XTP in VLSI – Protocol Decomposition for ASIC"*, Proceedings of the 15th IEEE COnference on Local Computer Networks, Minneapolis, Minnesota, September, 1990.

[6] W. Doeringer, *et. al.*, *"A Survey of Lightweight Transport Protocols"*, IBM Research Report, RZ 1980 (number 70152), May 17, 1990.

[7] James Moulton, *Final Report for Project 753*, ANSI document X3S3.3/90-201R, results of the study on Very High Speed Network Protocols, December 11, 1990.

[8] Ian Wakeman, *"Emerging High Speed Transfer Protocols"*, IEE Telecom Conference Proceedings, March 1991.

[9] Harry Rudin and Robin Williamson, editors, *Protocols for High-Speed Networks*, North-Holland, 1989.

[10] G. Chesson, *XTP/PE Design Considerations*, in *Protocols for High-Speed Networks*, [9].

[11] Robert Sanders, *XTP Tutorial*, University of Virginia Computer Science Department.

[12] Gerard J. Holzmann, *Design and Validation of Computer Protocols*, Prentice Hall, Englewood Cliffs, NJ 07632.