XTP/PE Design Considerations†

Greg Chesson
Silicon Graphics

Abstract

The **X**press **T**ransfer **P**rotocol, XTP, is an approach to protocol design for next generation networks. The **P**rotocol **E**ngine, PE, is a hardware architecture for implementing network protocols and system interfaces using VLSI techniques. The two projects are related: XTP is designed with a PE in mind. This paper presents some of the ideas and goals in XTP.

Introduction

XTP [1] represents a compromise between issues of bandwidth, latency, protocol complexity, VLSI constraints, transport services, system interfacing, and speculation about future network evolution. The general goals are to demonstrate feasibility of Gbit/sec operation, to fit in a VLSI environment, and to provide adequate functionality for distributed system applications.

Bandwidth

Network interfaces can be based on software plus specialized hardware for the link level functions, and can be effective at the 10 Mbit/sec level. As network bandwidths from 100 Mbit/sec to 2 Gbit/sec become available it is necessary to apply hardware resources to more than just the link level.

It is important to consider the number spaces defined by the protocol. For example an 8-bit or 16-bit sequence number is of little use on a 1 Gbit/sec channel, assuming that the sequence value is used to number bytes. All of the numbers in XTP are 32-bits except those that are clearly unaffected by bandwidth or other operating parameters of high performance media. There is also a provision for a 64-bit version of the protocol and a simple translation between 64-bit and 32-bit frame formats.

High bandwidth networks may be more susceptible to deep congestion than slower networks. This would happen because of larger host and network populations and greater traffic generating capacity at hosts. The onset of congestion will be more rapid than on slower networks because the offered load can reach extreme levels in shorter time periods than on low bandwidth networks. It is doubtful that such networks can avoid congestion by schemes that depend on detecting congestion at the endpoints of a connection and then "backing off" the offered load. In XTP the network assigns a maximum do-not-exceed bandwidth, or packet rate, to each unique path through the internet. The assignment is made and adjusted by the interior gateways and switches. The rate assignment mechanism operates in conjuction with the path finding mechanism.

Hardware Implementation

XTP has a goal of processing packets in real-time; i.e. to complete all protocol processing for an inbound or outbound packet in an amount of time equal to the transmission time of the packet on the media. This goal along with the high bandwidth target forces implementation technique to be considered during the protocol design process.

At 100 Mbits/sec the transmission time of a 128 byte packet is about 10 microseconds. At 1 Gbit/sec the time for a 1280 byte packet is about 10 microseconds. It is expected that many packets would be larger than 128 bytes or 1280 bytes in these two cases, but we take 10 microseconds as a useful target.

The creation of VLSI circuits is a fairly routine process and can be applied to network interfaces. Most protocols and system interfaces are too complex to be completely "hardwired". Therefore only a fraction of the network interface can be implemented in VLSI, or specialized protocol designs are needed, or perhaps both.

† Prepared for IFIP WG6.1/6.4 workshop on *Protocols for High-Speed Networks*, May 1989.

The 100 Mbit/sec performance level can be reached with a combination of RISC processing elements and custom circuits. In this kind of design the custom circuits provide a high speed data path for network traffic and the RISC elements support the more complex state and interface processing. This approach can be applied to all protocols, not just XTP, with varying degrees of success.

The long term goal for XTP is to achieve a compact VLSI representation of the complete design. This may require further evolution of the protocol, but some important groundwork has been established.

Protocol Complexity

A simple protocol fits easily into VLSI circuits. One example is the stop-and-wait protocol that retransmits on a timeout or negative acknowledgement and which incorporates no other control or addressing or multiplexing. Although such a protocol would have some utility, it was felt that XTP should have greater capability.

XTP is not the simplest protocol, nor is it the most complex. It is a minmax compromise: the most useful set of functions consistent with a simplified implementation. The core of XTP is a lightweight protocol based on the header address and sequence number plus bitflags in the trailer. This is sufficient to route and deliver data reliably. A simplified implementation is achieved by having fixed-size headers and trailers, avoiding options that add complexity, and ensuring that time-critical components in the performance data path are as simple as possible. Complexity is added outside the performance path, either above XTP or in parallel.

In addition to the core protocol there are numerous other other bitflags, fields, and modes. The important thing to know is that these facilities are manipulated by the the protocol or application layer **above** XTP. The XTP state machine does not calculate when to set the end-of-message bit or when to enable a special operating mode. These and other decisions are made above XTP, either in an application or in the "glue" layer between XTP and the layer using XTP.

XTP is designed to decompose into a number of parallel state machines. Examples are the flow machine, the error machine, the address translation/lookup machine, the open/close machine, the rate control machine, and the system interface machine to name a few. All networks incorporate such machines into their design, but none that we know of devote attention to execution in parallel. This technique opens the way to implementing high speed protocol subsystems using a number of lower speed components operating in parallel.

For a simple example we can consider using two RISC machines plus a custom VLSI data path as a protocol subsystem. One RISC handles all duties involving traffic between the subsystem and the network. The other RISC handles all traffic between the subsystem and the host. A single RISC would have to be powerful as well as very good at context-switching and interrupt handling to perform as the equal of two slower RISC engines that did not have to derail as often.

High performance Protocol Engine designs are planned which incorporate as many as eight RISC engines plus other specialized circuits packaged in a small number of chips: two RISC engines per chip. XTP, while easily implemented in software as a single thread process, is designed to decompose into this parallel environment.

The 80/20 Principle

Consider a software protocol implementation operating in a single-cpu workstation environment. The cpu cycles associated with moving information between the workstation and the network can be divided into two classes: *protocol-related* code for parsing and generating PDU's and executing the protocol state machines, and *non-protocol related* code for managing buffers, copying data, computing checksums, operating the network media hardware, managing timers, and interfacing to the operating system and applications. In many cases the protocol-related overhead is roughly 20% of the total networking overhead, leaving about 80% of the overhead attributable to non-protocol processing tasks.

This suggests that hardware assistance for networking should concentrate on the non-protocol-related tasks since they dominate. This is certainly reasonable, and is the only course of action when working with established protocol standards. For this reason a significant portion of the protocol engine design is targeted at the non-protocol-related tasks.

It is interesting to study the effect of protocol design on non-protocol-related overheads. For example it is possible to use check functions with less overhead. It is possible to organize the frame design to reduce data movement and to simplify the parsing and generation of protocol control fields. These techniques and others are employed in XTP to reduce non-protocol overheads.

Frame Design

Each XTP packet contains a fixed-size header and fixed size trailer. In between the header and trailer is either variable-length information segment containing user data or a control segment containing protocol state information.

XTP header	**information segment**	**XTP trailer**

XTP header	**control segment**	**XTP trailer**

Processing for inbound packets must begin with identifying the packet, associating it with a local object such as a socket or port, referencing the protocol state for the socket, and steering data in the packet to a destination. The XTP header is sufficient to get the processing started as a packet arrives at the protocol subsystem.

The XTP header contains four 32-bit words: *cmd, key, route,* and *seq.* The *cmd* word indicates packet type, version number, bitflags, and a data offset field. The *key* and *route* fields provide address and routing information. The *key* value is an endpoint connection identifier denoting a connection, or *context*. The *route* value selects a path through a router or switch. The *seq* field numbers the first information byte of each data packet as in TCP.

Early versions of XTP had only trailers, but that caused extra processing delays. Later versions of XTP had trailers only on packets containing an information segment. Packet processing became simpler when all packets contained a trailer.

Alignment

XTP is organized to minimize data movement caused by misalignment. The points of alignment in an XTP frame are at the start of the XTP header, beginning of the data or control segment, and at the beginning of the trailer. Padding is added where needed so that each of these alignment locations is on a 64-bit boundary. This eases software overheads for 32 and 64-bit computers. For hardware implementations it means that the header and trailer components are nicely aligned as they travel through a hardware data path thus simplifying circuitry for processing on input or generation on output.

A user buffer address may not be well-aligned to the word width or memory controller line size of the host computer. In order to avoid extra data movement or byte permutations in a DMA data path, an offset field within the XTP *cmd* word indicates the size in bytes of a possible gap between the beginning of the information segment and the actual start of user data within the segment. The gap is transferred intact to the receiving system where it can be processed in an implementation-dependent way: the gap can be eliminated by extra data movement or passed intact to receiving software that must be prepared to make an adjustment.

The tradeoff being made here is PDU size versus data movement and structure reference overhead. The choice has been to add some overhead bytes of media overhead. At bandwidths of 10 Mhz or less the choice seems to be unimportant, but appears to be a worthwhile consideration at 100 Mhz and above.

Check Functions

Protocols such as TCP and TP4 incorporate additive checksums over the packet data and control information. This is intended to provide a check across router elements which cannot be verified with a link-by-link crc function. XTP must provide a check function for these same reasons.

The TCP check function requires a 1’s complement add operation per 16-bits of data. The TP4 check function requires two add operations per byte. These functions must be implemented at the data rate of the media or at the data rate of the host DMA channel, whichever is faster. Although it is possible to calculate additive check functions at gigabit speeds, the cost is considerable.

It was considered important to select a check function for XTP that could be implemented with a small amount of logic in a parallel data path operating at host bus speeds. The choice is a 32-bit rotating XOR function which detects all odd-bit errors and misses only those even-bit errors characterized separated by exactly 1024 bits. This function in hardware takes the form of a 32-bit register with XOR inputs. It can be implemented in wider data paths that are a multiple of 32. In software it takes the form of a sequence of 32-bit XOR and ROTATE instructions and is considerably faster than add-based procedures.

Receiver Responses

Many protocols require the receiver to ACK data or to open an allocation window without stating an algorithm. Implementors must provide heuristics for receiver responses and then tune their systems. It is difficult to tune a receiver to respond well to different traffic distributions, for instance bulk transfer and asynchronous interactive traffic.

XTP provides a bit in the trailer for requesting a control packet from the remote receiver. The control packet contains the state of the receiver with respect to the information flow including error map, allocation window, delivery status, and other items. Except for error cases where a receiver may autonomously generate advisory control messages, receiver responses are controlled by the peer transmitter. The transmitter is thus able to adjust the rate and timing of responses from the receiver to suit the application traffic.

Routing and Rate Control

High speed media are evolving in two directions. One trend is toward so-called *fast packet switch* technology incorporating high-speed digital switches connected by point-to-point trunk circuits. The other trend is toward faster local area networks interconnected with various kinds of bridge and gateway systems.

Fast packet switch networks are based on virtual circuits over symmetric paths. Prototype systems have demonstrated multi-gigabit traffic carrying ability. The systems may damage packets in transmission, but packets are not duplicated, reordered, or inordinately delayed. The error model at the edge of a fast packet subnet is that of simple data loss.

Current gateway-based internets may replicate and reorder packets and have been known to hold packets for long time periods before forwarding to a destination. The error model at the edge of an internet is complex. Error control in transport for such a network is correspondingly more complex than for the fast packet switch model.

The routing strategy for XTP was chosen so that a hybrid subnetwork incorporating both LAN and switch-based technology could function smoothly; i.e. so that XTP can work well with both kinds of hardware. The idea is to organize gateways/routers so they operate more like the fast packet switches.

The *route* field in the XTP header has a zero value on the local net. When an initial connection-opening packet, called a FIRST packet, arrives at a gateway from a source (which may be a gateway) on the local net, the gateway parses the network destination address contained in the information segment of the packet. An exit port is determined and the gateway generates a *route* tag representing the triple: source, destination, and exit port. This route value is returned to the source. Subsequent non-FIRST packets from the source contain the assigned route value and are forwarded quickly by the gateway to the exit port. If there is a path through several gateways, the *route* value changes at each gateway.

The *route* field as described is similar to the *channel* numbers or *slot* numbers assigned to virtual circuits in hardware packet switch systems. It is anticipated that a gateway-like system can easily interface between an XTP LAN and a packet-switching system. At the boundary between domains the gateway would assign *route* tags that are actually channel numbers to a switch connected to the gateway.

XTP's routing strategy has additional properties. Once a source computer has obtained a *route* value, all packets sent with that value will be routed to the same destination through the internet. This means that the source system can transmit additional datagrams or create new connections to the destination host without waiting for the *route* assignment/exchange with a gateway.

One issue that must be dealt with is the management and retention of *route* values. Route values must not be reused unless they are inactive. The problem is in knowing when all the contexts using a route have finished. This is believed impossible to manage without help in the form of control messages between routers and host systems.

The creation of a new route requires a handshake with a gateway system. XTP gateways assign a maximum packet rate to each new route. The packet rate is returned to the source computer along with the new *route* value in a control packet. The rate is applied to all traffic from the source computer along the route. This has several beneficial side-effects. First, if the source system generates a large load on a particular route by initiating more connections, those connections must share the assigned rate thus avoiding a surge at the gateway. Second, if a source system wants to find a new path through the net and obtain a new or additional rate, it can do so. Third, the gateway may regulate assigned rates upward or downward in response to the observed load and available resources. The protocol does not specify how the gateway should count or assign resources. Fourth, a rate-based system uses fewer network packets than a scheme that achieves rate control with a moving window flow control scheme. The difference in network bandwidth may be negligible, but the difference in workload for the gateway or other system to generate window allocations would be noticeable in large scale systems. Fifth, the space required in an XTP router to administer rate control in this manner is related to the number of active source/destination pairs through the router. This is a much smaller number than the number of active connections which would be the case if the protocol architecture required intervention on a connection basis.

Context Management

XTP is based on a full-duplex virtual circuit model. A virtual circuit is also called a connection. The state variables at one end of a connection are referred to collectively as a *context*. A context is created when a Send or Receive command is posted to the network subsystem. The unique connection identifier, or *key*, is selected when a context is created on output. A sending context becomes half-open when the first packet is sent and becomes fully open, or *active*, when a response is received from the peer receiver. At this time the protocol subsystem obtains an initial estimate of round trip time.

Aliasing problems can arise if active keys are duplicated. The problems are avoided with a combination of techniques and conditions. The basic technique is a three-way handshake initiated by one side or the other or possibly both sides of a connection. The basic conditions are supplied by the XTP routing model which preserves message order. Thus it is not possible, after completing a three-way close handshake, for "old" packets to appear from the internet and be recognized as aliasing a new context.

If XTP is used over an IP layer which does not have the same properties as the XTP routing model, then contexts and context keys must be aged before reuse. The necessary timer value is a multiple of the maximum packet lifetime in the internet which could be minutes, hours, or days depending on circumstances.

The three-way handshake in XTP is designed to gracefully terminate contexts and account for all data to be delivered through subsystem queues. The close state machine uses three signaling bits in the trailer field and 9 bits in the context to keep track of things. There are four major traversal paths through the close machine. First, is the familiar graceful close. Second, is a not so graceful close where a context can be released before the queues drain at the other end of the network. Third, is the datagram close where a connection is opened and then shutdown with two end-to-end messages. Lastly, there is an abort path which circumvents the intermediate steps but does not guarantee data delivery. In any of these cases extra messages are generated from a timer in case of transmission or delivery error.

Context management in XTP is organized around a fast open for low latency and a three-way handshake for closing. The three-way handshake can be replaced by a two-way handshake in certain circumstances. The maintenance of *key* consistency is simplified by the router model. It is expected that high performance networks will be consistent with the switch-oriented router model. Significant space and timer overheads can accrue when this is not true.

Stream Syntax

Delta-T provides B and E symbols to mark the beginning and ending points in a data stream. The XNS protocol provides and end-of-message, EOM, bit as well as an 8-bit data stream type in every packet. Without such facilities higher level services and protocols would have to introduce another layer of parsing within the information segment. XTP provides similar mechanisms.

There is a BTAG bit in the header and an ETAG bit in the trailer which flag the beginning or ending portion of a packet. There is also an EOM bit that signifies an end-of-message mark in a stream. If marked with BTAG, the first 64 bits of the packet are interpreted as a "tag" associated with the rest of the data. ETAG similarly marks the last 64 bits of the packet. This provides sufficient code space for cooperating endpoints to exercise a variety of signalling, control, and encapsulation techniques.

There is a SORT bit in the header that indicates the presence of real-time scheduling data at the beginning of the information segment. This is one of the facilities provided for adapting XTP to the environment of real-time control systems.

Reservation Mode

In many protocol systems an allocation window is maintained as a receiver heuristic. XTP operates in this familiar manner and also provides "reservation mode" for bulk transfer. In this mode a bit in the header informs the receiver to provide an allocation window that represents only the amount of space available in locked down application receive buffers. On very high speed networks it seems necessary to be able to constrain a transmitter in this way, even at the possible expense of end-to-end delays. The reason is that an unconstrained transmitter with a large allocation window can quickly cause queue overflows. The reservation technique guarantees that the queues can be drained into receive buffers.

Datagrams

Datagram service is provided in three ways. Since a high performance implementation of XTP should be able to create contexts in real time, a datagram can be represented as a short connection. Such datagrams are reliable since the error control mechanism of the connection is in use. The error controls can be disabled by a bit in the header. In this case the behavior is similar to a connectionless service. Lastly it is practical and useful to define a datagram service over XTP connections. We are currently defining a convergence layer for operating a UDP protocol over XTP.

Summary

This paper has presented an overview of some of the techniques employed in the XTP design to reduce overheads associated with protocol implementation. Some of these techniques are syntactic, like the frame design and checksum. Other techniques are algorithmic, like the router model. It is believed that these and other techniques will be needed to fully utilize the promise of very high performance networks.

Acknowledgements

Silicon Graphics has provided support and encouragement for the XTP/PE project since 1986. Key understandings have been contributed by Vernon Schryver, Andrew Cherenson, Brendan Eich, Jim Hamstra, Bruce Carneal, Sandy Fraser, Dave Presotto, Dick Watson, Al Whaley, Doug Jenson, Dave Marlow, Steve Anderson, Ed Hotard, and Mark Cohn. The support and encouragement of the member companines of the Protocol Engines Technical Advisory Board has been an inspiration and key factor in XTP development. These companies include Ameritech, Apollo Computer, Ardent Computer, Artel, Boeing, Brooktree, DY-4, IBM, Intergraph, Interphase, Martin Marietta, Masscomp/Concurrent, Mentat, Silicon Graphics, Unisys, and Xerox.

References

[1] XTP Protocol Definition, Revision 3.3, Protocol Engines, Inc., Santa Barbara, Ca, 12 December 1988.