Communications for the NASA Space Station

Alfred C. Weaver and Robert Simoncic
Computer Networks Laboratory
University of Virginia
Charlottesville, Virginia 22903

ABSTRACT
The NASA Space Station is a truly international effort, and therefore its communications systems must conform to established international standards. For that reason NASA is requiring that each Network Interface Unit implement a full suite of ISO protocols which will operate over redundant FDDI networks. However, NASA is understandably concerned that a full ISO stack will not deliver performance consistent with the real-time demands of Space Station control systems. Therefore, as a research project, we are investigating whether the Xpress Transfer Protocol (XTP) is a suitable candidate for use alongside a full ISO stack. This paper describes our initial plans for implementing XTP and for comparing its performance to ISO TP4. We present performance graphs for the basic datalink service which underlies the protocol stack.

1. BACKGROUND OF THE SPACE STATION PROJECT

The NASA Space Station, now scheduled for launch in the mid-1990s, will be a distributed system supporting a global network of international science, technology, and commercial users. The Space Station Information System (SSIS) is responsible for providing communications services between end users; users may be men or machines, and they may be located on-station or on the ground. SSIS must interface interoperably with data streams from many different sources and must transport a wide variety of data types and data rates, while at the same time remaining sufficiently flexible that it can accommodate the technology changes which will certainly occur over the Station's expected 30-year lifespan.

One important component of SSIS is the Data Management System (DMS) which provides the hardware resources and software services which support the data processing and communications needs of the Station's systems and payloads. DMS will provide a common operating environment and human-machine interface for operation and control of the Space Station. DMS has defined a set of services which it will provide to the user:

- file transfer and access in space, on the ground, and within international partners' networks
- on-board virtual terminal system
- remote job entry service
- real-time and non-real-time telemetry and telecommand

- electronic mail
- SSIS-wide, application-to-application messaging service
- local and remote access to on-board databases
- connectionless multicast distribution of messages (ancillary data)
- globally-unique, location-independent names
- global naming authority

NASA's initial performance requirements for DMS define three grades of service. Grade I specifies a connection-oriented (i.e., virtual circuit) service in which every message is guaranteed to be delivered in order and without duplication. The underlying network must operate with a bit error rate of \( 10^{-12} \) or better. Grade II is a datagram service, in which messages may occasionally be lost, or duplicated, or delivered out of sequence. This service must support a bit error rate of better than \( 10^{-9} \). Grade III is a poorer datagram service, with the same specifications as Grade II except that the bit error rate may rise to \( 10^{-5} \). Each of these grades of service fills a particular need. For example, grade I service would be required for program upload, while grade III would be sufficient for real-time voice.

In addition, performance requirements for message latency are being established for four classes of messages: background, normal, isochronous, and emergency. Latency is the time elapsed from the moment the application process requests message transfer until the message arrives at its destination if it is on the same DMS network, or until it arrives at the appropriate gateway if it is bound for an international partner's module or for the space-to-ground RF links. While the exact latency requirements for the four message classes have not yet been finalized, they are expected to be in the range of tens of milliseconds.

2. THE SPACE STATION LAN

An important system design consideration for DMS was the selection of the Space Station LAN. The initial candidates were the IEEE 802.3 contention bus, the 802.4 token bus, the 802.5 token ring, ANSI's Fiber Distributed Data Interface (FDDI), and SAE's High Speed Ring Bus (HSRB). The first three were rejected early on and the remaining two were studied in detail. Both FDDI and HSRB share some attractive characteristics:
FDDI was finally selected because its specification was more mature and because there was the promise of significant commercial support, especially in the form of chip sets, for the standard.

The Space Station baseline design now calls for FDDI as the backbone LAN to interconnect the guidance, control, and navigation subsystems, the file servers, the life support subsystems, the display consoles, the communications and tracking subsystems, and the gateway to the RF uplink/downlink. At each such interface, there is a Network Interface Unit (NIU) which provides communications services to the attached host. The NIU implements all seven layers of the OSI reference model and provides the physical connection to the FDDI backbone.

Because the Space Station is truly an international project, with active participation from Europeans, Canadians, and Japanese, the communications system must be interoperable over multiple vendors and heterogeneous space and ground computer systems. This has led NASA to adopt the ISO OSI services and protocols as being the only hope for achieving international interoperability. But everyone recognizes that therein lies a dilemma. Since OSI was developed in the environment of national, packet-switched, wide area network communications, its design emphasis was on interoperability, not performance. Thus one challenge is to develop hardware and software which can communicate via the ISO protocols and yet achieve the throughput and latency requirements needed for SSIS --- this challenge has been undertaken by Honeywell for the ground-based testbed and by IBM for the flight-qualified system. A second challenge is to determine whether or not another, more advanced technology such as the Xpress Transfer Protocol (XTP) is suitable for use on Space Station and whether it has performance attributes consistent with the needs of real-time systems. This second challenge has been undertaken by the Computer Networks Laboratory at the University of Virginia.

3. WHY XTP?

NASA is understandably concerned that a full ISO protocol stack will not perform adequately for a real-time system. Various published measurements of ISO protocol performance [Janetzky87, Heatley88, Strayer88a, Strayer88b, Svoboda89] suggest that, however adequate ISO protocols may be for general purpose use (e.g., file transfer), they are not generally considered adequate for real-time control systems.

With funding from the United States Naval Ocean Systems Center, we investigated a number of protocol alternatives for real-time systems [Strayer88c], ranging from full seven-layer protocols (MAP, TOP) to transport protocols (ISO TP4, VMTP, XTP, GAM-T-103) to MAC-layer protocols (FDDI, HSRB). While none of these was perfect, we determined that XTP was the closest match to the needs of real-time systems. Those unfamiliar with XTP may refer to [Chesson87a, Chesson87b, Chesson88].

XTP is potentially a much higher performance protocol than ISO TP4 or TCP. Quoting from the XTP Protocol Definition:

"The functional design for XTP arises from the needs of contemporary and future distributed systems. Existing protocol systems, e.g., TCP and ISO TP4, do not meet these needs. In addition to well-known performance and complexity issues, the most cited problem is that they provide only a "traditional" reliable stream service, whereas distributed systems need reliable real-time arbitrary-sized datagrams. This reflects the need of distributed systems for remote procedure calls, rapid request/response operations, and transaction-based file servers."

Unlike TCP or TP4, XTP is designed from the outset to be implementable in VLSI hardware, and its technology is designed to scale from 10 Mbit/s (e.g., Ethernet) to 100 Mbit/s (e.g., FDDI) to 1 Gbit/s networks. XTP is equally applicable to LANs, MANs, and WANs.

4. XTP IMPLEMENTATION PLANS

Our XTP design has only begun, so its description herein is necessarily incomplete. Figure 1 shows our proposed architecture.

Clients are users of the XTP service. They may be application processes in their own right (such as real-time control programs), or they may be other communications services (such as the ISO session layer).

The User Interface is a set of communication services provided by XTP to the user. At the moment there is no official XTP service definition, so we are developing our own. One idea under consideration is that the user interface may look like a control block in which the user specifies an operation (e.g., send data), a size, and a pointer to the head of a buffer chain. By configuring the buffer as a buffer chain, we believe that we can achieve memory economy for short (e.g., control) messages while still supporting arbitrarily large messages (e.g., files). An XTP user wanting to transmit, say, a 10 MB file will have to segment it somewhere, so the linked segment approach provided by the buffer chain provides a conceptually simple way for the user (or its operating system) to accomplish it.

The XTP State Machine would be our code implementing XTP version 3.4 as per the definition in [XTP89]. We are making every effort to code XTP as the Finite State Machine (FSM) which it is. The advantages to us are three-fold: (1) we would expect our best performance to emerge from a FSM implementation; (2) the FSM representation will ease our eventual conformance testing against the XTP Software Reference Model (SRM); (3) a FSM implementation would allow our code to run as a microcontroller (similar to the Protocol Engine concept, but without the custom VLSI).

A Context is an XTP term meaning an active connection or datagram. A Context Record is created for each new connection or datagram processed. The XTP state machine manipulates the context record to modify the status of a connection.
The Context Controller is our code to multiplex and demultiplex multiple contexts (connections) across a single Logical Link Control (LLC) interface. It is a small finite state machine which performs two primary operations:

1. When receiving data it performs a lookup (using the key field in the XTP header) to identify the proper context to be associated with the incoming message. If the connection has already been established, then the proper context record is found and the message is enqueued for that context. If this is a datagram or the first message of a connection, then a new context record is created. Various algorithms are under consideration to provide a fast lookup service (balanced trees, hash tables, etc.) but no decision has yet been made.

2. When transmitting data the FSM makes a departure from standard protocol coding practice. When a message to be transmitted (either data or acknowledgement) is received from the XTP state machine, the context controller will indeed enqueue the message for transmission, but will not frame the message or otherwise initiate the transmission process. Instead, it continues to collect messages for transmission until it receives a signal from the Signal Handler advising that the transmitter is idle and needs work. Only then are the messages passed to the LLC for framing and transmission. The advantage of this technique is that we are still prepared to transmit at every legitimate opportunity (i.e., whenever the transmitter becomes idle) and we will transmit all enqueued messages (data and acknowledgements) in the minimum number of frames. For example, if two sequential messages are received for the same destination, then ordinarily two separate acknowledgements will be framed, enqueued, and sent to the original transmitter. If in our system the acknowledgement for the first message has not been transmitted by the time the acknowledgement for the second message is prepared, then we will combine the two acknowledgements into one, thereby reducing the number of frames transmitted.

The Signal Handler is another small finite state machine which further decouples the transmit and receive processes from the packetization process in the LLC. When the LLC receives, deframes, error checks, and accepts a packet it signals the Context Controller that one or more packets are in the receive queue. Only when the Context Controller is ready to process those received messages are they physically retrieved from the common buffer pool. When the LLC transmitter goes idle, the Signal Handler signals the Context Controller that the transmitter needs more work. All accumulated data and acknowledgements are then moved to the LLC transmitter where they are examined and then packetized into the fewest possible packets.

The Signal Handler operates a circular queue of buffers on both its transmit and receive sides. An interesting question is how best to operate the receive buffers. Suppose that the receive buffers are temporarily full and a new message arrives. With classical protocols the last message is simply lost due to buffer starvation (this can happen with or without flow control, depending upon the timing). In our design we want to assure that there is always an available receive buffer, but if the receive buffer pool is full, which buffer should be reused? If the data being sent is strictly sequential, then we should delete the most recently received message (because the oldest message is most important since it affects the sequencing and acknowledgements). If, on the other hand, this is datagram-type traffic (e.g., sensor readings or effector updates in which the most recent information is most valuable), then we should delete the oldest message in the queue. But the kind of data being transmitted is, in general, unknown, and "reading the mail" to determine what type of data a packet carries is not generally considered good procedure either. Thus, receiver buffer management is a timely research issue, and we will experiment with various strategies to determine what works best.
Similarly, it is unknown whether the receiver queue, described above as a simple circular buffer, should actually be a priority queue. Our studies of priority systems thus far [Peden87, Peden88a, Peden88b] have shown that medium access priority strategies, for example the "timed token" of IEEE 802.4 and FDDI or the "priority reservation" of IEEE 802.5 and SAE HSRB, make only small differences in overall message latency in the general case. However, processing priority within the protocol stack makes a great deal of difference, leading us to believe that our receiver should actually implement a priority queue. So this, too, is a fruitful research area.

5. PERFORMANCE

Our initial implementation of XTP will be on an ALR FlexCache which contains a 25 MHz Intel 80386 processor, 128 KB of 25ns cache memory, and 4 MB of 40ns RAM. To determine the potential performance of this equipment, we implemented a standard IEEE 802.2 datalink service for class 1 (connectionless) messages. These performance measurements are valuable for two reasons: (1) the Network Interface Unit must contain a full ISO stack, which in turn includes the IEEE 802.2 datalink service. Thus a full ISO stack implemented on this equipment will run no faster (and in fact will be considerably slower) than these measurements. (2) Although XTP itself does not require a separate datalink frame, the amount of message handling required by XTP will be similar to that required for 802.2. Thus our datalink measurements will also provide a realistic upper bound for the eventual performance of XTP itself.

Our datalink service was implemented in three different ways:

(1) using programmed I/O on a 4 Mbit/sec Proteon ProNET-4 token ring network, which complies with the IEEE 802.5 token ring standard and which presents a firmware interface for the datalink services. The resulting performance curves are labelled "P4/pio" in the graphs.

(2) using direct memory access on a 10 Mbit/sec Proteon ProNET-10 token ring. The ProNET-10 is a simpler protocol than IEEE 802.5 and thus has a simpler interface. For the ProNET-10 we provide all the 802.2 framing services in software on the host processor. The performance curves are labelled "P10/dma".

(3) using programmed I/O on a 10 Mbit/sec Proteon ProNET-10. This is the same as (2) above except that messages are moved from the host to the front end processor via programmed I/O instead of direct memory access. The performance curves are labelled "P10/pio".

Figure 2 documents the transmitter throughput which can be achieved by a single host. Transmitter load is measured in Kbit/sec and is plotted as a function of message size. Note that a single station can generate nearly 4 Mbit/sec on a 10 Mbit/sec token ring when using 2000-byte messages. The same station can generate 2 Mbit/sec when interfacing to the IEEE 802.5 token ring.

![Figure 2. Transmitter Load (Kbits/sec) vs. Packet Size (bytes)](image)

Figure 3 shows transmitter throughput from a different perspective, this time in packets/sec. Real-time networks are often more concerned with message rate than with message volume (i.e., throughput), so the packets/sec measurement is important as a gauge of the responsiveness of the network. The measurements presented for a message length of zero bytes are particularly enlightening. A data frame whose message length is zero is the minimum frame possible and therefore reflects pure overhead. No "real" message frame can be faster than one with no data to process. Thus this measurement gives us an upper bound on message rate — 4260 packets/sec when using ProNET-10 and programmed I/O. Since real-time networks often carry short messages, the measurement for frames with 100-byte messages is fairly realistic. Our system supports 2380, 1185, and 636 packets/sec for the P10/pio, P10/dma, and P4/pio systems respectively.
Figure 3.
Transmitter Load (packets/sec) vs. Packet Size (bytes)

Figure 4 reveals another important aspect of real-time networks — latency. These measurements result from putting the application process in the host into an infinite loop and then measuring the time between successive messages. For a 100-byte control message, we see that the P4/pio system can emit a message every 420 µsec. Note that the P10/dma system has better performance (i.e., lower latency) than the P4/pio system for packets shorter than 700 bytes, but worse performance for longer packets. This points out that DMA operations are not always the best choice for speed.

Figure 4.
Transmitter Delay (ms) vs. Packet Size (bytes)

For control systems we must also be concerned with the end-to-end delay as shown in Figure 5. End-to-end delay is the elapsed time from the moment the message is ready in the transmitting application process until the message is received in the destination application process. This measurement includes all operating system overhead, message copying, transfer from the host to the front end processor across the CPU backplane, queueing time in the front end processor, network access time, network transmission time, and all the symmetric delays in the receiver. Since this measurement is application process to application process, this is a true measure of the system performance which a real application process would observe. For the P10/pio system, end-to-end delay varies from 680 µsec for 100-byte messages to 6.6 ms for 2000-byte messages.

Figure 5.
End-to-End Delay (ms) vs. Packet Size (bytes)

Figure 6 presents one more system statistic — host processor idle time. If the host CPU is fully utilized, then additional protocol cannot be executed without degrading system performance. Our measurements show a range of idle times, from 81% idle in a P4/pio system to 16% idle in a P10/dma system for 100-byte messages. The P4/pio system has more idle time than either P10 system because it has a protocol processor (the Texas Instruments TMS380 token ring chip set) on the network adapter board, whereas the P10 systems rely on the host processor for all protocol operations at and above the Logical Link Control layer. Within our P10/pio system, idle time ranges from 20% for 100-byte messages to 40% for 2000-byte messages. These numbers show that some CPU capacity remains for executing more complex protocols before absolute performance begins to degrade.

Figure 6.
Host Processor Idle Time (ms) vs. Packet Size (bytes)
6. CONCLUSIONS

Previously cited performance measurements suggest that conventional ISO protocol implementations may not adequately support the throughput and latency goals of the FDDI-based Space Station Data Management System. Purely as a research project, we are investigating whether XTP is a suitable alternative to ISO TP4. Our initial performance data for a datalink service implemented on a 25 MHz Intel 386 and a 10 Mbit/sec Proteon ProNET-10 token ring suggests that we can approach a sustained loading of 4 Mbit/sec (using 2000-byte messages) from a single NIU. Our end-to-end latency (from transmitter's user memory to receiver's user memory) ranges from 680 µsec for 100-byte messages to 6.6 ms for 2000-byte messages.

The challenges which remain include:

1. replace the IEEE 802.2 LLC with the XTP transport service. At least for short messages which do not require segmentation, the goal is to have the transport service operate at nearly the same speed as the underlying datalink service.

2. design and implement a suitable user interface for XTP. Two levels of user interface are contemplated, one which exposes the power and internal workings of XTP and is therefore complex, and one which hides those details and is therefore simple to use. The former will be needed by the real-time control systems whereas the latter is preferable for routine file transfer, messaging, and electronic mail.

3. replace the IEEE 802.5 and ProNET-10 networks with an FDDI network. While FDDI has 10 times the capacity of the ProNET-10, it is unlikely that any single station will see greater throughput than we have already observed. The ProNET-10 device interface is exceptionally simple, which makes it fast. The FDDI interface is known to be much more complex.

4. show by demonstration that upper-layer ISO protocols, in particular the session layer, can operate over XTP as well as over TP4. Since upper-layer ISO protocols are a firm requirement, no replacement for TP4 can be contemplated unless it interoperates with the OSI upper layers.

7. REFERENCES


338


