The Delta-t Transport Protocol: Features and Experience

Richard W. Watson

Lawrence Livermore National Laboratory
P.O. Box 808
Livermore, CA 94550

Abstract

With the advent of new high performance networks and distributed systems there is renewed interest in transport protocol designs that can support both request/response and stream styles of communication. Delta-t is a transport protocol designed to meet such goals. Delta-t's main contribution is in the area of connection management, where it achieves hazard free connection management without explicit packet exchanges. This paper reviews Delta-t's features, connection management more generally, and outlines some implementation lessons useful for high performance networks.

1. Introduction

At the Lawrence Livermore National Laboratory (LLNL) we have developed an integrated network and distributed operating system architecture that we call the Livermore Integrated Network Computing System (LINCS) [12, 26]. LINCS was designed to integrate a wide range of heterogeneous micro to supercomputer systems. It has been implemented as the native multiprocessing operating system on our Cray XMP and YMP systems, called NLTSS [10, 23, 24, 26], and as a guest on various flavors of Unix, VMS and other systems.

LINCS is a capability based, multitasking, message passing, client/server architecture. Capabilities are protected by servers against forgery through encryption, and are just ordinary data kept in application memory space and sent in messages. LINCS defines a number of standard server supported abstract object interfaces (e.g., file, process, directory, clock, account, etc.), an interprocess communication model, and a set of communication protocols from the link to application levels [12, 21, 25]. Many of the protocols will be replaced by national or international standards when appropriate ones are developed and are vendor supported.

Early on in this project we recognized that we needed a high performance transport protocol that could efficiently support both the request/response transaction style of communication needed for client/server interactions and the stream style of communication needed for terminal sessions and bulk data transfer.

Our list of requirements was the following:

1. Minimum packet exchange for request/response transactions.
2. High throughput bulk data transport and other stream services.
3. Flow control without polling for reliable zero window opening.
4. Error control of lost, damaged, duplicated, and out-of-sequence packets.
5. Large and flexible name space for transport end points.
6. Message boundary preservation.
7. Secure communications.

In 1977, when the project began, the only available general purpose transport protocol was TCP [13]. TCP was not widely supported by vendors at that time and required excessive connection management packets to exchange a request and response. It also failed to meet other of the requirements above. Therefore, we decided to reexamine the connection management problem and meet our other requirements.

We did not want a special-purpose transport protocol that made limiting assumptions about the topology or error properties of the network or was specially tailored just for the request-response style communication [1, 20]. We wanted a general-purpose transport protocol that could meet all our needs in a general network topology. From this effort the Delta-t protocol emerged [11, 21]. Delta-t's design goal was to allow complete requests or replies to be sent using two packets in the usual case, one packet for the request or reply and one for its Ack. No other packets should be required for connection opening or closing. Because an Ack can acknowledge more than one data packet, bulk data transfer can be achieved using less than one Ack per data packet, using delayed Acks.

Delta-t has had several implementations and has been operational about seven years. The sections below outline what we have learned from this work useful in protocol design and implementation for high performance networks. Delta-t's main contribution to high performance, or lightweight, protocol design is in the area of connection management [18, 19]. Delta-t's connection management mechanisms can be used in both connection oriented link level and transport protocols. We focus here on transport protocol design. It is our observation that connection management is still a troublesome area in transport protocols because they often contain connection management hazards or do not clearly define the network topology and error assumptions that would make them safe. Even in high performance networks these hazards must be considered.

Section 2 outlines the features of Delta-t. Section 3 reviews the main requirements and mechanisms of connection management that must be dealt with or used by any transport protocol. Section 4 outlines other lessons we have learned.

2. Outline of Delta-t's Features

The functionality of Delta-t is split into a connectionless network level protocol and a transport level protocol. The former handles those services provided by datagram routing nodes (including routing software in the ends) and the latter those services provided just by the ends. Here we describe them together as if Delta-t were a single protocol. Delta-t can be implemented on other connectionless network protocols such as DARPA's IP or ISO's CLNP.

Naming

Communication takes place between ports on processes. A process consists of a memory address space, a port address space, and one or more lightweight threads of control, called tasks, that share the memory address and port address spaces. Ports are identified by 64 bit addresses unique over the network. These addresses can be location independent generic addresses, or location specific hierarchical physical addresses. The former can be mapped through a name server to the latter.

Security

All packets carry two security related fields, a four bit protection label (unclassified, confidential, secret, etc.) so that a mandatory security policy can be enforced by all nodes of the network, and a 64 bit random number, called a stream number described below. Besides checking the protection label, routing nodes check source addresses at the boundary between different administrative or protection domains to assure that they are legitimate in the source domain. This check prevents masquerade by processes outside the next domain to which the packet is to be routed.

The stream number, which is generated by the application level, is in effect a communication stream capability establishing the sender's right to communicate the data or Ack packet. It is also used as a transaction identifier for synchronizing requests with their associated replies. A stream is a unidirectional flow defined by the triple (destination port, source port, stream number). Higher level communication primitives allow send and receive specifics, where all three members of the triple are specified, or send and receive anys, where one or more members of the triple can be any value. Stream numbers and support for send and receive specifics and anys have been very useful in designing flexible, access controlled higher level communication structures between two or more parties not initially known to each other.

Data Units

Delta-t supports an alphabet visible at the transport interface consisting of bits (0,1) and the symbols B, E, W. Higher level protocols define how the latter are used. The position in the stream of the B, E, and W symbols is visible at the transport interface. Generally B and E delimit the beginning and ending of some high level unit such as a message or monolog [25]. The W can be used by a sender to indicate a point of advisory wakeup when it has sent enough information for the receiver to process usefully. Bits were chosen as the fundamental unit of error and flow control rather than bytes because at the time Delta-t was designed we had 36 and 60 bit machines as well as machines that were byte oriented. The cost of orienting Delta-t to bits has been low, specifically 3 bits in sequence number and flow control window fields.

Flow Control

Delta-t supports sliding window flow control for the same reasons that sliding window flow control was chosen for TCP [9]. It also supports a rendezvous-at-the-sender mechanism for reliably handling the zero window opening hazard that results from possible lost or out-of-sequence acknowledgements [21]. In Delta-t when the receiver advertises a zero window to the sender, or knows the sender is facing a zero window, it will later send a reliable zero window opening control packet. The rendezvous-at-sender mechanism relieves the sender from polling when facing a zero window, as it must do, for example, in TCP [13]. When the sender sends an E or W symbol both receiver and sender assume that the window, as viewed by the sender, is zero because most implementations will mark a buffer as complete when either of these symbols arrive.

Our experience, like that of others, has been that the interaction of flow control and buffer management strategies is a troublesome area and a cause of implementation performance problems, such as generating extraneous packets [6, 7, 20]. Therefore, were we to start again we would want to examine alternate flow control mechanisms such as rate-based flow control appearing in newer protocols [2, 4, 5].

Assurance

Delta-t uses sequence numbers on bits and the B, E, W symbols to protect against loss, duplication, and out-of-sequence data. Damaged data is detected by an optional checksum. Delta-t supports hazard free connection management, without connection management packet exchanges, in a network with an arbitrary mesh topology and the possible errors listed above. The term connection is used to mean that time during which state information is maintained at each communicating end. Connection opening refers to reliably establishing this state and connection closing refers to reliably and unambiguously deallocating this state. Because we believe connection management is a somewhat subtle area and needs to be handled properly by newer high performance protocols, we review this area in more detail in the next section.

Delta-t's connection model supports the view that logically there are permanent error and flow controlled connections between all possible streams, each defined, as mentioned above, by the triple (destination port, source port, stream number). When a connection is in a default state, no state need be retained. Delta-t's timer-based connection management mechanism, outlined below, automatically recognizes when a connection is or is not in the default state and allocates and deallocates connection state, without connection management packet exchanges.

Delta-t's connection management mechanism is conceptually straightforward. For a given stream the sender and receiver maintain connection state and a send-timer and a receive-timer, respectively, when data are being exchanged. The initial values for these timers and the rules for their operation assure that the opening and closing requirements listed in the next section are met. These rules are specified and derived in references [11, 21]. Simply stated, the receive-timer rules assure that state is maintained long enough so that all old duplicates are detectable, assuring reliable connection opening, and that the node will not accept packets for a safe interval on crash recovery. The send and receive timers together assure that state is maintained long enough to guarantee a graceful close and that acceptable sequence numbers are generated and accepted. The send-timer rules assure that sequence numbers will not be reused, even in the face of a node crash, until all data or Ack packets with a given sequence number have expired.

A time-to-live field in packet headers is decremented during packet routing, retransmission, and acknowledgement to assure that the lifetime of a Delta-t packet is bounded. A related packet header field allows the sender to communicate to the receiver what the initial value of this lifetime was for the oldest sequence number in the packet, thus enabling the receiver to set its timers properly and set the time-to-live field of corresponding Acks correctly.

3. Connection Management

All transport protocols, whether for wide area or high performance local area networks, must protect against the connection management hazards introduced by lost, duplicated, or out-of-sequence packets, unless special network error properties can be assumed in the protocol design. All of these connection management hazards are frequently not protected against in existing or proposed transport protocols, particularly in a general multipath mesh network. The main contribution of the design of Delta-t was to demonstrate the fundamental role of timer mechanisms in safe connection management mechanism and to develop a pure timer-based connection management scheme.

There are three basic mechanisms used in connection management: 3-way handshake, unique-connection-identifiers, and timer. Even with the first two, timer-based mechanisms are required for hazard free connection management. This is because handshakes or unique-connection-identifiers by themselves cannot meet all of the following connection management requirements:

General

G1: An identifier of an information unit used for error control (or any other service) must not be reused while one or more copies (duplicates) of that unit or its Ack are alive.

G2: The error control information being transmitted between each end must itself be error controlled.

G3: If the crash of an end can cause it to lose its state, then appropriate crash recovery mechanism must assure the other requirements are met.

Connection Opening

O1: If no connection exists, and the receiver is willing to receive, no duplicate packets from a previously closed connection should cause a new connection to be established and duplicate data to be accepted, unless the operations represented by the data are known to be idempotent.

O2: If a connection exists, then no packets from a previously closed connection should be acceptable within a current connection.

Connection Closing

C1: No packet from a previous connection should cause an existing connection to close.

While the ambiguity eliminated by the next two graceful close requirements is still possible if an end node crashes or the network partitions, we want to limit ambiguity to exactly these events and not have ambiguity introduced by the operation of the protocol. Unless C2 and C3 are met the sender could be left in the unnecessarily ambiguous position of not knowing if a receiver received all data that was sent. The designers of the ISO transport protocol chose to meet these requirements at higher protocol levels [14].

C2: A receiving side should not close until it has received all of a sender's possible retransmissions and can respond to them.

C3: A sending side should not close until it has received acknowledgement of all that it has sent. In particular it should allow time for an acknowledgement of its final retransmission, if needed, before reporting a failure to its client program.

Below we briefly review some of the issues in meeting the above requirements using 3-way handshake, unique-connection-id, and timer-based mechanisms. Our recommendation to transport protocol designers is that they do a complete case analysis in the face of all the hazards that can occur in the environment for which they are designing (e.g., lost, duplicated, or out-of-sequence data and control packets in the general mesh network) to assure these requirements are met.

First, we need to discuss bounding error control identifier lifetime as needed to meet requirement G1. It is of course equally true that the lifetime of identifiers used for flow control, secure communications or other service must also be bounded if they are different from those used for error control. This need to bound identifier lifetime in units of time rather than routing hops is now widely recognized and most network level protocols include a time-to-live field in their headers. This maximum-packet-lifetime (MPL) is to be enforced at the routing level (including the ends). MPL can also be enforced between gateways across a network lacking packet aging services [16].

Bounding MPL in the routing network is a valuable and necessary service, but by itself is not sufficient because the error control identifiers live in the end nodes as well. On the sending side there are send queuing delays and there is the retransmission interval, R, that the identifier may be held and then be resent into the network. On the receiving side there is the queuing interval from the receipt of a packet until transport protocol processing begins and the interval, A, until an acknowledgement is issued during which an identifier can live. We assume that time spent on packet queues in the end nodes is included in the routing level packet aging process. Therefore, an error control identifier, such as a sequence number or unique-connection-id [4] or unique-port-identifier [2, 3], could live 2MPL + R + A, if MPL is initialized to the same value on all packets containing a given error control identifier.

The identifier lifetime bounding process needs to take the R and A intervals as well as MPL into account. For example, in Delta-t implementations, successive retransmissions enter the network preaged by TTL = T(I) - T(R), where TTL is the value set in the time-to-live field of a packet, T(I) is the maximum time-to-live of the first transmission of the oldest identifier in a packet and T(R) is the time since the first transmission of the oldest identifier to a given retransmission. Similarly Acks enter the network preaged by TTL = T(I) - T(A), where TTL is as above, T(I) is the initial time-to-live of the received identifier and T(A) is the time to generate the Ack. T(I), or what we have called Delta-t, is chosen by the sender as a function of reasonable routing network MPL, known R, and expected A.

Determining a reasonable R and A is difficult if application level libraries implement the protocol, because both R and A are functions of process scheduling, which is affected by system load and application priority. The result is that if the protocol is implemented in application level libraries, even in a high performance local area network environment, a T(I), in the 60 second or longer range, may be required. This need to implement protocols, under certain circumstances, in application level libraries and its impact on protocol design and implementation is often ignored.

For safe connection management, state information must be held for particular periods of time. The state held and the period of time it is held is dependent on the protocol and on how particular connection management mechanisms are used by a given protocol. Analysis of several existing or proposed transport protocols over the years has shown that they contain connection management hazards because state is not being held sufficiently long. Because we are in an active period, when new protocols are being designed and possibly standardized, for high performance networks, it seems useful to briefly review connection management using 3-way handshakes, unique-connection-ids, and timers. We assume a general multipath topology, where lost, duplicated, and out-of-sequence packets are possible.

3-Way Handshake

3-way handshakes can be used for either connection opening, closing, or both. For example, TCP uses 3-way handshakes for both opening and closing. VMTP optionally can use a 3-way handshake on opening, and XTP uses a 3-way handshake for closing [2, 4, 13].

VMTP normally uses timer mechanism for closing, which when combined with unique-connection-ids (implemented as unique entity addresses) does not normally require a 3-way open. However, if the end is recovering from a crash or discards state earlier than would be safe, then a 3-way handshake open is required.

XTP uses unique-connection-ids for opening to meet condition O2 and C1, but requires, not yet specified, timer mechanism to meet O1. To meet requirements C2 and C3 it uses a 3-way close to try to discard state as quickly as possible. Unfortunately this handshake still does not meet O1.

A 3-way handshake opening protocol meets requirement O1 by having the receiver in effect ask the sender on each connection opening request "do you really mean it or is this a duplicate". It requires timer mechanism, or a unique-connection-identifier (e.g., in TCP the careful choice of initial sequence numbers in effect create unique-connection-ids), and associated careful choice of the size of the identifier space and possible control on the rate of identifier generation in order to meet requirements G1, G3, O2 and C1 [18,19]. Requirement G2 can be met if the connection management "open" and "close" control flags are included in the sequence space or are otherwise error protected.

Timers are required in a 3-way close protocol, such as TCP, to meet conditions C2 and C3 [18, 19]. This need for timers in some 3-way close protocols results because the last packet containing a close flag may also contain data or the Ack of the close packet could also Ack data. The Ack of the data and close could be lost. In any protocol the last message cannot be critical, because it is not Acked. If the sender of the last close-ack closed immediately after emitting the close-ack (implicitly Acking data as well), and its close-ack got lost, then the other side would timeout and retransmit the unAcked data and close flag. The closed side on receiving the retransmitted information can only generate a Nak or reset packet indicating it was closed. The receiver of the reset would then not be able to distinguish this case where there was successful delivery of data, but a lost Ack, from the case of a crash or network failure with lost data. Therefore, the sender of the last close-ack (the receiving side of requirement C2) must wait an interval in order to assure it can respond to all its correspondents retransmitted close packets.

Alternatively another type of Ack packet could be used to in effect Ack the close-ack. When such a packet was received, state could be safely discarded because if this packet were lost no ambiguity would result as no data is involved.

A giveup timer is required to meet requirement C3. This giveup timer results because the sender must allow time for its last retransmission, time for an Ack to be generated, and time for the Ack to return before reporting failure to high layers. If a problem is reported before a giveup interval, the user application process may create an unnecessary duplication or the application process may unnecessarily enter an involved error recovery procedure to deal with what appears as a possible partner crash or network failure.

Therefore, in order to meet requirements C2 and C3 protocols using a 3-way handshake close mechanism need timer mechanism for hazard free operation.

Unique-connection-ids

A unique-connection-id based protocol, meets requirements G1, G3, O2 and C1 if it can guarantee generation of unique-connection-ids for each new connection, through appropriately picking a unique-connection-id space size and possibly limiting rate of identifier use. Unique-connection-id protocols require stable storage or clock mechanism in the face of crashes in order to assure uniqueness. To meet G1, G3, O2 and C1, a unique-connection-id cannot be reused, after a connection has closed or after a sender or receiver recovers from a crash, for a period long enough to guarantee all packets with that identifier have expired. The receiver must maintain state under timer control after a connection closes until all duplicates, including retransmissions have expired in order to meet requirement O1. Even if a 3-way handshake close is used, state must be maintained for an interval to meet requirement O1. The timer periods for holding state to meet O1 depend on the protocol details. G2 is met, for example, by error control of the combination of unique-connection-id and sequence number. Unique-connection-ids cannot assure a graceful close and therefore must be combined with timer or 3-way handshake mechanism or both for a safe closing meeting C2 and C3.

Timer

A timer-based protocol is one that maintains state under timer control in order to meet the requirements G1, G3, O1, O2, C1, C2, and C3, for example as outlined for Delta-t in Section 2. Requirement G2 is met because, in a protocol such as Delta-t, there are no opening or closing flags that need to be protected and the ordinary sequence number mechanism for data is all that is required. There are many possible timer-based protocols. While timer protocols are very simple in concept, determining the correct timer values and rules for timer operations is protocol dependent and somewhat subtle. The timer values and rules for Delta-t needed to meet the above requirements are derived in references [11, 21]. Timer considerations for VMTP are discussed in references [2, 3].

In a timer-based protocol, the receiver maintains error control or other service identifier state under send- and/or receive-timer control. The receive-timer is refreshed each time a new identifier is accepted, the connection is closed, or as the result of some other rule required by a given protocol. The interval of the receive-timer is chosen so as to guarantee that all sender retransmissions and other duplicates will be recognized in order to meet requirements O1, O2, C1, and C2. If a receiver crashes and loses state, it must wait a specified interval before accepting packets to meet G1, G3, O1, O2, and C1, in order for retransmissions and duplicates of identifiers sent before the crash to expire.

The sender maintains identifier state under send-timer control long enough to guarantee that it can generate acceptable unique identifiers and that all data sent or resent have a chance to be acknowledged as per requirements G1 and C3. If a sender crashes, then it must wait a specified interval on recovery in order to meet requirements G1 and G3.

Summary

The connection management design tradeoff facing protocol designers is that of trading off extra packet exchanges, when 3-way handshakes are used, or state retention, when timer mechanisms are used. Even with a 3-way close, timer mechanism may be required unless the 3-way close occurs after all data has been sent and Acked. Unique-connection-id protocols must be combined with one or both the 3-way handshake or timer mechanisms for hazard free connection opening and closing.

Because connection management is a subtle issue we would like to see protocol descriptions or specifications explicitly state the network topology and error assumptions they are designed to deal with and explicitly show how they meet the connection management requirements given above. Further, it would help in determining the correctness of the protocol if any timer rules for state retention or for reuse of unique-connection-identifiers be explicitly stated and be related to the MPL, R, and A identifier lifetime factors discussed earlier.

4. Lessons Learned with Delta-t Applicable to High Performance Protocol Design and Implementation

This section outlines briefly our experience with the design and implementation of Delta-t applicable to transport protocols for distributed systems and high performance networks.

1. A general purpose transport protocol for an arbitrary mesh network can be designed and implemented that meets both the need for efficient request/ response (minimum packet exchange, low latency) and stream (high throughput) oriented styles of communication. Packet exchanges can be minimized for request/response communication by the use of timer-based connection management. Newer flow control techniques may offer improvements over sliding-window-based flow control for high throughput requirements, although recent optimization work with TCP indicates sliding-window-based flow control can also yield good performance [7, 15].

2. Use of a random number, at least 64 bits in size, as a communication stream capability, and supporting send and receive interface semantics can simplify higher level protocols involving secure third party communication or communication between parties that do not initially know each other's addresses, but do know the stream capability.

3. Performance is overwhelmingly an implementation rather than a transport protocol design issue [6-8, 15, 20]. One must clearly distinguish between the complete transport layer implementation, which may span user and system levels, and the transport protocol implementation. The transport layer implementation contains the application to operating system interface, data copying, queuing, transfer status, buffer management, operating system service interfaces, and lower protocol layer interface code that is largely independent of the transport protocol. In addition, significant time can be spent in device drivers.

   The transport protocol is just a subroutine. In our experience with Delta-t implementations only 5-15% of the time to send or receive a packet is in the transport protocol processing. Similarly only 10-25% of the code is for the protocol algorithm. Our experience has shown that the main cost to send and receive packets, and in fact often the number of packets exchanged, is heavily influenced by transport protocol independent issues such as:

   - the application (user) level to operating system interface design, and its impact on the number of context switches required and on data copying and buffer management strategy (e.g., does the system buffer data to be sent, is the data for possible retransmission kept in application or system space, what application level data structures need updating for send or receive status),

   - buffer and memory allocation strategies and their impact on data copying and the number of data packets and Acks generated (e.g., is space preallocated in packet buffers for packet headers in order to minimize copying, are user buffers aggregated into packet buffers to minimize the number of data packets sent?),

   - network driver and lower level protocol interfaces and implementations (e.g., the nature of the host and network interface I/O architecture, such as number of interrupts required to handle a packet; whether or not there is a link level protocol and how it is implemented),

   - acknowledgement and flow control strategies and their interaction with buffer management affect smoothness of data flow, the number of packets exchanged, and the size of data packets sent (e.g., is each packet acknowledged or are Acks delayed in order to reduce their number; how is overflow handled, is there multiple buffering, does an end advertise window values based on system or user buffers, are window advertisements accurate),

   - lightweight tasking (e.g., the use of lightweight tasking in our implementations has been effective for implementation structuring and multiprocessing, but more expensive for monoprocessing, due to extra context switching overhead),

   - design decisions made in order to develop an implementation portable across a range of operating systems (e.g., the inability to use system specific facilities such as mbufs in B.s.d 4.3 Unix systems, the use of additional interfaces for portability, and the fact, for example, that the system interface efficient on a Cray may not be efficient on a SUN or VAX or vice versa).

4. Supporting a proprietary protocol in a heterogeneous environment is costly. For an efficient implementation we needed to place the transport level in the kernel of several vendor's operating systems and even in several versions of a single vendor's operating system. We tried to do this as a standard device driver in order not to have to make kernel modifications. Structuring the implementation as a device driver added overhead. In addition, we found that avoiding kernel modifications was not completely possible, even across different vendor's UNIX systems, because of UNIX or vendor specific limitations, such as the lack of support for a kernel lightweight tasking mechanism, or an inability, via a switch table accessible from a utility routine, to route incoming packets to the appropriate driver based on a link level protocol field or link level address. This need to make even small kernel modifications (e.g., changing a half a dozen instructions) has created high support cost due to:

   - the source code licence cost and the long delays to get kernel source code,

   - delays caused by having to learn vendor kernels and to make and debug modifications, even if simple,

   - operating system quality assurance, distribution and support,

   - possible conflicts with prime or third-party vendor conventions on kernel data structure usage.

   Striving for portability has also meant that we could not take advantage of certain optimizations available to vendors in their kernel protocol implementations. For example, not all UNIX vendors support B.s.d. mbufs; therefore for portability we needed a separate portable buffer management mechanism and this required extra data copies from our portable buffer structure to mbufs and vice versa. Data copying is one of the most expensive operations and must be minimized for optimal performance [7].

This lesson indicates to us that only highly optimized, vendor supported implementations are likely to realize the full advantage of a given protocol at reasonable cost.

5. Given the performance cost of implementing transport protocol mechanism in software, high performance may be aided by optimizing aspects of protocol design such as minimizing options, using fixed sized word aligned fields, and placing checksums in a trailer; by placing transport and lower level layer mechanism in hardware; and by developing the I/O and operating system architectural mechanisms necessary to allow direct DMA from application memory through transport level processing chips to the network [17]. That is, since most of the time is spent in transport layer non-protocol specific processing, new system and network I/O architectures are needed to simplify or offload it. Without this capability the application to operating system, operating system processing, and conventional device driver overhead will continue to dominate the performance, independent of improvements in transport protocol design [7].

We can summarize these lessons by stating that while it is important to keep developing improved protocol mechanisms, the main areas requiring work are improving: implementation techniques; application-to-network, operating system and I/O architectures; and getting incremental protocol improvements through the standardization process in a more timely fashion.

Acknowledgement

I wish to acknowledge the many contributions of the following people at various stages of Delta-t design and implementation: Jed Donnelley, John Fletcher, Jed Kaplan, Dan Nessett, Alex Phillips, Lansing Sloan, Dave Wiltzius, and Rich Wolski. This work was performed by Lawrence Livermore National Laboratory under contract number W-7405-Eng-48 under auspices of the U.S. Department of Energy.

References

1. Birrell, A. D., and Nelson, B. J. "Implementing Remote Procedure Calls", ACM Trans. Comput. Syst. 2, 1, Feb. 1984 pp. 39-59.

2. Cheriton, D. R., "VMTP: A Transport Protocol for the Next Generation of Communication Systems". In Proceedings of the SIGCOMM '86 Symposium on Communications Architectures and Protocols (Stowe, Vt., Aug. 5-7). ACM, New York, 1986, pp. 406-415.

3. Cheriton, D. R., "VMTP: Versatile Message Transaction Protocol Specification," Computer Science Dept., Stanford University, 22 February, 1988.

4. Chesson, G., "XTP Protocol Definition" Revision 3.3, Protocol Engines, Inc., 12 December, 1988.

5. Clark, D. D. Lambert, M. L., and Zhang, L. "NETBLT: A Bulk Data Transfer Protocol", DARPA Network Working Group RFC 969, Network Information Center, SRI International, Menlo Park, CA, December 1985.

6. Clark, D. D., "Window and Acknowledgment Strategy in TCP," Internet Protocol Implementation Guide, Network Information Center, SRI International, Menlo Park, CA, Aug. 1982.

7. Clark, D. D., V. Jacobson, J. Romkey, H. Salwen, "An Analysis of TCP Processing Overhead," IEEE Communications Magazine, June 1989, pp. 23-29.

8. Clark, D. D., "The Structuring of Systems Using Upcalls", In Proceedings of the 10th ACM Symposium on Operating Systems Principles, Orcas Island, Wash., Dec. 1-4, ACM, New York, 1985, pp. 171-180.

9. Clark, D. D., "The Design Philosophy of the DARPA Internet Protocols", Proc. ACM SIGCOMM '88, Computer Communications Review, Vol. 18, No. 4, Aug. 1988, pp. 106-114.

10. Donnelley, J. E., "Components of a Network Operating System", Computer Networks 3, 1979, pp. 389.

11. Fletcher, J. G., and Watson, R. W., "Mechanisms for a Reliable Timer-based Protocol", Computer Networks, 2, North-Holland, Amsterdam, The Netherlands, 1978, pp. 271-290.

12. Fletcher, J. G., "Introduction to LINCS", Available through the Lawrence Livermore National Laboratory Computer Center as Chapters 1-12, Lawrence Livermore National Laboratory, Tentacle Apr.1982 to Mar. 1983.

13. Information Sciences Institute. DOD Standard Transmission Control Protocol. Information Sciences Institute, Marina del Ray, CA, September 1981, Available from Network Information Center, SRI International as RFC 793.

14. International Standards Organization. Information processing systems--open systems interconnection-transport protocol specification. International Standards Organization, ISO/DIS 8073, Rev., ISO/TC 97/SC 16WG 6, June 29, 1984.

15. Jacobson, V., "Congestion Control and Avoidance", Proc. ACM SIGCOMM '88 Symposium, ACM, Aug. 88, Stanford, CA.

16. Sloan, L., "Mechanisms that Enforce Bounds on Packet Lifetimes", ACM Trans. Comput. Syst. 1, 4, Nov. 1983, pp. 311-330.

17. Kanakia, H., Cheriton, D., "The VMP Network Adapter Board (NAB): High Performance Network Communication for Multiprocessors".Proc. SIGCOMM 88 Symposium, ACM, Aug. 88, pp. 175-187.

18. Sunshine, C. A. and Dalal, K. K., "Connection Management in Transport Protocols", Computer Networks 2, 4/5, Sept./Oct. 1978.

19. Watson, R. W., "Timer-based Mechanisms in Reliable Transport Protocol Connection Management", Computer Networks 5, North-Holland, Amsterdam, The Netherlands 1981, pp. 47-56.

20. Watson, R. W. Mamrak, S. A., "Gaining Efficiency in Transport Services by Appropriate Design and Implementation Choices", ACM Trans. on Computer Systems, Vol. 5, No. 2, May 1987, pp. 97-120.

21. Watson, R. W., Delta-t protocol specification. UCID-19293, Lawrence Livermore Laboratory, Livermore, CA, Apr. 1983.

22. Watson, R. W., and Fletcher, J. G., "An Architecture for Support of Network Operating System Services". Computer Networks 4, North-Holland, Amsterdam, The Netherlands 1980, pp. 33-49.

23. Watson, R. W., "Notes on Operating System Requirements for the Next Millennium," Proceedings, Cray User Group Meeting, Minneapolis, April 1988.

24. Watson, R. W., "The Architecture of Future Operating Systems," Proceedings Cray User Group Meeting, Tokyo, September 1988.

25. Watson, R. W., "LINCS Session, Presentation, Common Application Protocols", Lawrence Livermore National Laboratory, December 2, 1982.

26. Watson, R. W., "Working Notes: Motivation Goals, and Development Strategy for NLTSS", Lawrence Livermore National Laboratory, July 1987.