The Multidriver: A Reliable Multicast Service Using the Xpress Transfer Protocol

JOHNSON GRANT
IN-62-CR
115098
P.24

Bert J. Dempsey
John C. Fenton
Alfred C. Weaver

Digital Technology

August, 1990

Cooperative Agreement NCC 9-16
Research Activity No. SE.31

NASA Johnson Space Center
Engineering Directorate
Flight Data Systems Division

Research Institute for Computing and Information Systems
University of Houston-Clear Lake

INTERIM REPORT

The RICIS Concept

The University of Houston-Clear Lake established the Research Institute for Computing and Information Systems (RICIS) in 1986 to encourage the NASA Johnson Space Center (JSC) and local industry to actively support research in the computing and information sciences. As part of this endeavor, UHCL proposed a partnership with JSC to jointly define and manage an integrated program of research in advanced data processing technology needed for JSC's main missions, including administrative, engineering and science responsibilities. JSC agreed and entered into a continuing cooperative agreement with UHCL beginning in May 1986, to jointly plan and execute such research through RICIS. Additionally, under Cooperative Agreement NCC 9-16, computing and educational facilities are shared by the two institutions to conduct the research.

The UHCL/RICIS mission is to conduct, coordinate, and disseminate research and professional level education in computing and information systems to serve the needs of the government, industry, community and academia. RICIS combines resources of UHCL and its gateway affiliates to research and develop materials, prototypes and publications on topics of mutual interest to its sponsors and researchers. Within UHCL, the mission is being implemented through interdisciplinary involvement of faculty and students from each of the four schools: Business and Public Administration, Education, Human Sciences and Humanities, and Natural and Applied Sciences. RICIS also collaborates with industry in a companion program. This program is focused on serving the research and advanced development needs of industry.

Moreover, UHCL established relationships with other universities and research organizations, having common research interests, to provide additional sources of expertise to conduct needed research. For example, UHCL has entered into a special partnership with Texas A&M University to help oversee RICIS research and education programs, while other research organizations are involved via the "gateway" concept.

A major role of RICIS then is to find the best match of sponsors, researchers and research objectives to advance knowledge in the computing and information sciences. RICIS, working jointly with its sponsors, advises on research needs, recommends principals for conducting the research, provides technical and administrative support to coordinate the research and integrates technical results into the goals of UHCL, NASA/JSC and industry.

RICIS Preface

This research was conducted under auspices of the Research Institute for Computing and Information Systems by Bert J. Dempsey, John C. Fenton and Alfred C. Weaver of Digital Technology. Dr. George Collins, Associate Professor of Computer Systems Design, served as RICIS research coordinator.

Funding was provided by the Engineering Directorate, NASA/JSC through Cooperative Agreement NCC 9-16 between the NASA Johnson Space Center and the University of Houston-Clear Lake. The NASA research coordinator for this activity was Frank W. Miller of the Systems Development Branch, Flight Data Systems Division, Engineering Directorate, NASA/JSC.

The Multidriver: A Reliable Multicast Service using the Xpress Transfer Protocol

Bert J. Dempsey, John C. Fenton, and Alfred C. Weaver

Department of Computer Science
Thornton Hall
University of Virginia
Charlottesville, Virginia 22903
(804) 924-7605
bjd7p@virginia.edu, jcf5a@virginia.edu, weaver@virginia.edu

Abstract

A *reliable multicast* facility extends traditional point-to-point virtual circuit reliability to one-to-many communication. Such services can provide more efficient use of network resources, a powerful distributed name binding capability, and reduced latency in multidestination message delivery. These benefits will be especially valuable in real-time environments where reliable multicast can enable new applications and increase the availability and the reliability of data and services. In this paper we present a unique multicast service that exploits features in the next-generation, real-time transfer[1] layer protocol, the Xpress Transfer Protocol (XTP). In its reliable mode, the service offers error, flow, and rate-controlled multidestination delivery of arbitrary-sized messages, with provision for the coordination of reliable reverse channels. Performance measurements on a single-segment Proteon ProNET-4 4 Mbps 802.5 token ring with heterogeneous nodes are discussed.

[1] The *transfer* layer incorporates the functionalities of the transport and network layers of the ISO OSI Reference Model into a single layer.

Table Of Contents

References

1. M. Ahamad, M. H. Ammar, J. M. Bernabeu-Arban and M. Khalidi, Using Multicast Communication to Locate Resources in LAN-Based Distributed System, *Proceedings of the 13th Conference on Local Computer Networks*, Minneapolis, Minnesota, 1988.
2. *FDDI Token Ring Media Access Control Standard*, American National Standards Institute, Feb. 1986. Draft proposed Standard X3T9.5/83-16, Rev. 10.
3. K. Birman and T. Joseph, Reliable Communication in the Presence of Failures, *ACM Transactions on Computer Systems* 5,1 (February 1987), 47-76.
4. J. Chang and N. F. Maxemchuk, Reliable Broadcast Protocols, *ACM Transactions on Computer Science* 2,3 (Aug. 1984), 251-273.
5. D. R. Cheriton and W. Zwaenepoel, Distributed Process Groups in the V Kernel, *ACM Transactions on Computer Systems* 3,2 (May 1985), 77-107.
6. D. Cheriton and C. L. Williamson, VMTP as the Transport Layer for High-Performance Distributed Systems, *IEEE Communications Magazine*, June 1989, 37-44.
7. G. Chesson, The Protocol Engine Project, *Unix Review*, September 1987.
8. D. Comer, *Internetworking with TCP/IP*, Prentice-Hall, Englewood Cliffs, New Jersey, 1988.
9. E. C. Cooper, Circus: A Replicated Procedure Call Facility, *Fourth Symposium on Reliability in Distributed Software and Database Systems*, 1984.
10. J. Crowcroft and K. Paliwoda, A Multicast Transport Protocol , *CCR 18*,4 (Aug. 1988), 247-256.
11. *The Ethernet: A Local Area Network — Data Link Layer and Physical Layer Specifications*, Digital Equipment Corporation, Intel Corporation, Xerox Corporation, November 1982.
12. D. T. Green and D. T. Marlow, SAFENET — A LAN for Navy Mission Critical Systems, *Proc. of the 14th Conference on Local Computer Networks*, Minneapolis, Minnesota, October 1989.
13. *IEEE Standard 802.2 Logical Link Control*, Institute of Electrical and Electronics Engineers, 1984.
14. *IEEE Standard 802.4 Token-Passing Bus Access Method and Physical Layer Specifications*, Institute of Electrical and Electronics Engineers, 1985.
15. *IEEE Standard 802.5 Token Ring Access Method and Physical Layer Specifications*, Institute of Electrical and Electronics Engineers, 1985.
16. *IEEE Standard 802.3 Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications*, Institute of Electrical and Electronics Engineers, 1985.
17. *Information Processing Systems - Open Systems Interconnection - Basic Reference Model*, International Organization for Standardization, Oct. 1984. Draft International Standard 7498.

18. T. Joseph and K. Birman, Reliable Broadcast Protocols, in *Distributed Systems*, S. Mullender (editor), ACM Press, 1989, 293-319.

19. M. F. Kaashoek, A. S. Tanenbaum, S. F. Hummel and H. E. Bal, An Efficient Reliable Broadcast Protocol , *Operating Systems Review 23*,4 (October 1989).

20. J. Kramer, J. Magee and A. Lister, CONIC: An Integrated Approach to Distributed Computer Control Systems, *IEE Proceedings Part E 130*,1 (January 1983), 1-10.

21. J. Lederburg and K. U. K., Towards a National Collaboratory, *Report of an Invitational NFS Workshop*, March 1989.

22. L. Liang, S. T. Chanson and G. W. Neufield, Process Groups and Group Communications: Classifications and Requirements, *IEEE Computer 23*,2 (February 1990), 56-66.

23. D. T. Marlow, Requirements for a High Performance Transport Protocol for Use on Naval Platforms, Revision 1, Naval Surface Warfare Center, July 1989.

24. J. F. McNabb and A. C. Weaver, A Real-Time Network Performance Monitor for Token Rings, *MILCOM 89*, Boston, Mass., October 1989.

25. *Xpress Transfer Protocol Definition: Revision 3.4*, Protocol Engines, Incorporated, Santa Barbara, California, July 1989.

26. B. Rajagopalan and P. McKinley, A Token-Based Protocol for Reliable, Ordered Multicast Communication, *Proceedings of Eighth Symposium on Reliable Distributed Systems* , Seattle, Washington , October 1989.

27. S. Ramakrishnan and B. Jain, A Negative Acknowledgement with Periodic Polling Protocol for Multicast over LANs, *IEEE INFOCOM 1987: The Conference on Computer Communications Proceedings*, San Francisco, California, April 1987.

28. R. Simoncic, A. C. Weaver, B. G. Cain and M. A. Colvin, SHIPNET: A Real-time Local Area Network for Ships, *Proc. of the 13th Conference on Local Computer Networks*, Minneapolis, Minnesota, October 1988.

29. W. Stallings, *Handbook of Computer Communications Standards, Volume 1: The Open Systems Interconnection (OSI) Model and OSI-Related Standards*, Macmillan, Inc., 1987.

1. Introduction

Multicasting refers to a communication facility for effecting delivery of a message to a well-defined set of destinations. Many modem networks, in particular Local Area Networks (LANs) conforming to the IEEE 802 standards ([14-16]), Ethemet ([11]), the ANSI FDDI standard ([2]), propagate frames such that all nodes on a frame's originating segment have the opportunity to capture it. Host interfaces support hardware filtering on group addresses, making machine-level multicast widely available in LANs. Recent research efforts have focused on integrating this underlying selective broadcast with the reliable services of peer protocols in the higher layers of the ISO OSI Reference Model ([17]). The resulting *reliable multicast* promises substantial efficiencies in network resource utilization and reduced latency for multidestination messages.

The need for multicasting arises naturally in a number of existing and emerging applications: resource location in a LAN ([1]), distributed databases ([3] [4]), industry process control ([20]), support for distributed operating system services ([19] [5]), replicated procedure calls ([9]), support for real-time command-and-control platforms ([23]), and collaborative development systems ([21]). Exploiting parallelism in delivery and message processing, multicasting may enable real-time applications whose message latency requirements cannot be met with unicast protocols. Applications such as resource location use a multicast to achieve a run-time binding between a logical set of destinations (the *multicast group*) and the current group membership. The run-time binding afforded by location independent addressing reduces the complexity in managing group communication and provides a high degree of service and data availability. Reliable one-to-many communication also opens up the possibility of synchronizing distributed processes without incurring the network-wide processing overhead and security problems inherent to broadcasting.

The networking community has recognized that, as the trend toward distributed systems continues to accelerate, multicast support within next-generation transport layer protocols will represent an important new functionality. First generation protocols (e.g. the DoD Transmission Control Protocol (TCP) [8] and ISO TP-4 [29]) did not anticipate the changes over the past decade in underlying network hardware, transmission speeds, and communication patterns that have enabled and driven the interest in reliable multicast. In this paper we present a unique multicast service that takes advantage of features in the next-generation, lightweight transfer layer protocol, the Xpress Transfer Protocol ([7]) (XTP).

As a modular part of the service interface routines to the University of Virginia implementation of the protocol (UVA XTP), our multicast facility, the Multidriver, exploits features in XTP in order to provide two multicast services, one unreliable and the other reliable. The Multidriver user receives a channel to which the user may submit either a complete message buffer or a data stream. Message boundaries are preserved, and a mechanism exists for synchronizing data delivery at the set of receivers. The Multidriver's reliable service uses as its primitive multicast transactions. It provides error, rate, and flow-controlled delivery in both the forward (request) and reverse (responses) channels. User-specified bounds control the number of multicast group members that may participate in an exchange and the number that must receive the request in order for the transfer to be considered successful.

The remainder of the paper is organized as follows. Section 2 outlines the basic architecture of UVA XTP and the function of *XTP drivers*. Section 3 describes the Multidriver design, service primitives, and control scheme. Section 4 examines related work from the literature. Section 5 presents some performance measurements, and Section 6 our conclusions.

2. UVA XTP Drivers

The UVA XTP architecture has a layered structure. The bottom layer represents the 802.2 Logical Link Control (LLC) ([13]) interface to the network. Above the LLC sits the XTP Engine, which performs protocol processing on XTP *contexts*, the structures that hold connection state information at an endpoint. (In UVA XTP each context must either be a transmit or a receive context.) At the highest layer reside *XTP drivers*. Drivers are special purpose modules that use the low-level interface to the Engine to implement an XTP service interface. Engine and driver communicate through shared memory in the context structures, a small set of C subroutines, and upcalls. XTP drivers handle decisions about retransmission, flow control, synchronization, and buffering in order that the Engine performs only the protocol processing common to all XTP users.

The separation of policy (driver) and mechanism (Engine) enables great flexibility in designing the user interface to UVA XTP. Drivers can be tailored to the communication needs of a particular application or class of applications. To facilitate driver development, we have implemented a set of primitives that are functionally modeled on UNIX system calls so as to provide a well-known user interface. From them XTP drivers have been written for several communication services, including file transfer, memory-to-memory transfer, and stream I/O. Driver primitives interoperate so that an application links with a driver library and includes only the code necessary for that application. The code fragment in Figure 1 illustrates the use of driver routines. It shows an application that reads characters from the network and displays them until the connection is closed.

```
main( )
{
  XTP_startup();
  if (xhandle = X_open("pipein","r",device)) < SUCCESS)
    { fprintf(stderr,"Unable to receive from network0);
      XTP_finish(-1); }
  while ((c=X_getc(xhandle)) > EOF)
    putchar(c);
  X_close(xhandle);
  XTP_finish(1);
}
```

Figure 1 — XTP Driver Primitives

3. The Multidriver

Our multicast facility is implemented as an XTP driver (the Multidriver) that provides the user with four primitives. For unreliable service, the user calls an initializing routine to set up state for a multicast transmit context. Data transfer through the context is then available using any of the driver routines. For reliable multicasting, the user calls an initializing routine that carries out a series of actions: (1) it sets up the transmit context; (2) it creates a user-specified number of receive contexts (*response contexts*); (3) it issues a connection set-up packet from the transmit context in order to establish the multicast (forward) connection; and (4) it monitors the contexts set up in (2) to ensure the establishment of some user-controlled number of connections between multicast group members and the response contexts. After these connections are made, the user can carry out a reliable multicast transaction with the set of receivers that have established response channels.

XTP provides a packet of type FIRST, which can carry user data as well as addressing information, to set up a connection. After a FIRST packet establishes the connection, the data source issues DATA packets. In a reliable unicast, the sending context determines when the receiving context will issue CNTL packets, which contain control information, by setting certain request bits in out-going DATA (or FIRST) packets. Within the byte-sequenced data stream of an XTP connection, out-of-band, or *tagged*, data can appear as the first 8 (BTAG header bit set) or the last 8 (ETAG header bit set) bytes of a DATA or FIRST packet. At the remote end, XTP passes up tagged data uninterpreted to the user ([25]). The Multidriver suppresses XTP's error control, e.g. the multicast transmitting context never sets status request header bits in a DATA packet. For reliable multicasting, the Multidriver manages its own control scheme by sending control information as tagged data, which can be multiplexed with user data, in both the forward and reverse channels.

3.1. Multidriver Design

The Multidriver design focuses on extending the unicast virtual circuit paradigm to a one-to-many connection. Implementation of this model requires solving synchronization and coordination problems not encountered in unicast protocols. Control information for the multicast connection must be efficiently and effectively collected at the multicast source and there coalesced into directives for the multicast transmit context.

Reliable one-to-many delivery implies the existence of some method for tracking the progress of a set of receivers. Otherwise, the multicast sender cannot provide reliable delivery since it cannot detect lagging or failed receivers. Rather than constructing its own method for handling the control flow from multiple data sinks, the Multidriver uses a well-defined mechanism already available within the Xpress Transfer Protocol — XTP connections. Unicast connections to response contexts create channels for driver-level control information as well as client data.

Laminating together XTP connections makes sense for a number of reasons. First, XTP supports rapid connection set-up and tear-down. An XTP FIRST packet can establish a connection and carry user data (as well as deliver tagged data). Connection tear-down involves a 2- or 3-packet handshake that is initiated by the final DATA packet in the transfer. Second, the mapping of individual receivers to their response channels takes place dynamically as incoming FIRST packets establish connections with response contexts; no prior coordination or management is needed. Finally, since control communication can be restricted to tagged data, the side channels enable bi-directional user data flows, i.e. multicast transactions. Reply handling in client/server interactions has been recognized as an important component of multicast communication in many classes of applications ([22]).

For many-to-one data flows, particularly within a LAN, the phenomenon of *network implosion* must be addressed. Implosion refers to the tendency of multicast receivers to synchronize the sending of their control packets in any transmitter-driven scheme. Synchronized transmission can result in bursts of traffic on the network and the inability of the multicast source's network interface to capture frames arriving back-to-back. Since the Multidriver supports the gathering of user-level responses from multicast group members, the problem of coordinating the reverse channels grows with the product of the amount of data in the reverse channel from each receiver and multicast set size. The Multidriver implements mechanisms that allow the multicast source to control network implosion. The administrator of implosion control policy, whether a human user or a management protocol, can use these mechanisms to determine the appropriate implosion control strategy. The synchronization issues involved with network implosion are highly dependent on system parameters. Hence the appropriate strategy is for the multicast communication facility to provide the user with parameters that can be tuned to the target environment.

The Multidriver design implements error, flow, and rate control for the multicast connection. The multicast source solicits control parameters from the set of receivers in the exchange. At the multicast source, after each receiver has responded, the Multidriver takes the minimum of the reported control parameter values and submits this information to the multicast transmit context in the form of an XTP CNTL packet. Since the multicast transmit context is unaware that CNTL packets are being manufactured from above (by the Multidriver) instead of arriving from below (off the network), protocol processing inside the XTP Engine takes place exactly as with reliable unicasting. In this way, control information from multiple communication endpoints is coalesced into directives for controlling the multicast transfer without the addition of extra checks within the transmit Engine.

Error control uses a go-back-N retransmission strategy as selective retransmission for multicast in a LAN environment (e.g. low latencies and relatively high bandwidths) seem unjustifiably complex. The Multidriver releases data in the transmit buffer as soon as arriving control information indicates that all receivers in the exchange have that data. Flow and rate control policies conform to the smallest values reported from the receiver group since faster transfer will only result in costly errors due to dropped packets.

3.2. Multidriver Service Primitives

X_Mopen(name, mode, device)

Depending on the value of `mode`, this routine either opens up a multicast context for unreliable multicast transmission or opens up a context for multicast reception. In the latter case, `X_Mopen()` opens a receive context that listens on the group address associated with `name`.

For unreliable multicast transmission, a transmit context is initialized such that the header bits for multicast (MULTI) and datagram transmission (NOERR) will be set in all FIRST and DATA packets issued from the context. Header bits requesting CNTL packets are guaranteed not to be set in any out-going packet. Otherwise, `X_Mopen()` performs the same state initialization as its unicast counterpart, `X_Open()`. `X_Mopen()` returns a handle of type XFILE that the user must use in driver calls to identify the opened XTP context.

The parameter `name` has a string value identifying a multicast group. The string is mapped internally to the addresses that identify the set of listeners that make up the multicast group. These addresses include the medium dependent hardware address, typically a group address, and the transfer layer address. The format of the transfer layer address depends on the environment since XTP supports multiple addressing modes. The parameter `device` is present since a single implementation of XTP can multiplex between multiple network interfaces.

X_MRopen(group, response, device, min, max, &xfiles[max])

`X_MRopen()` performs a series of actions. It sets up a transmit context to send to the multicast address to which `group` maps, and it initializes `max` receive contexts to listen on the address to which `response` maps. A FIRST packet is issued from the transmit context with tagged data containing the value of the parameter `response`, and a timer set. Upon expiration of the timer, `X_MRopen()` returns to the user with an error indication if fewer than `min` multicast set members have established connections with local response contexts. Otherwise, `X_MRopen` returns the user a XFILE handle to the multicast transmit context and places a XFILE handle in the array `xfiles[]` for each active response context, up to a limit of `max`.

X_MRclose(xfile, &xfiles[])

`X_MRclose()` closes a reliable group transmit context, `xfile`, and its associated response contexts, `xfiles[]`.

X_MRreply(xfile)

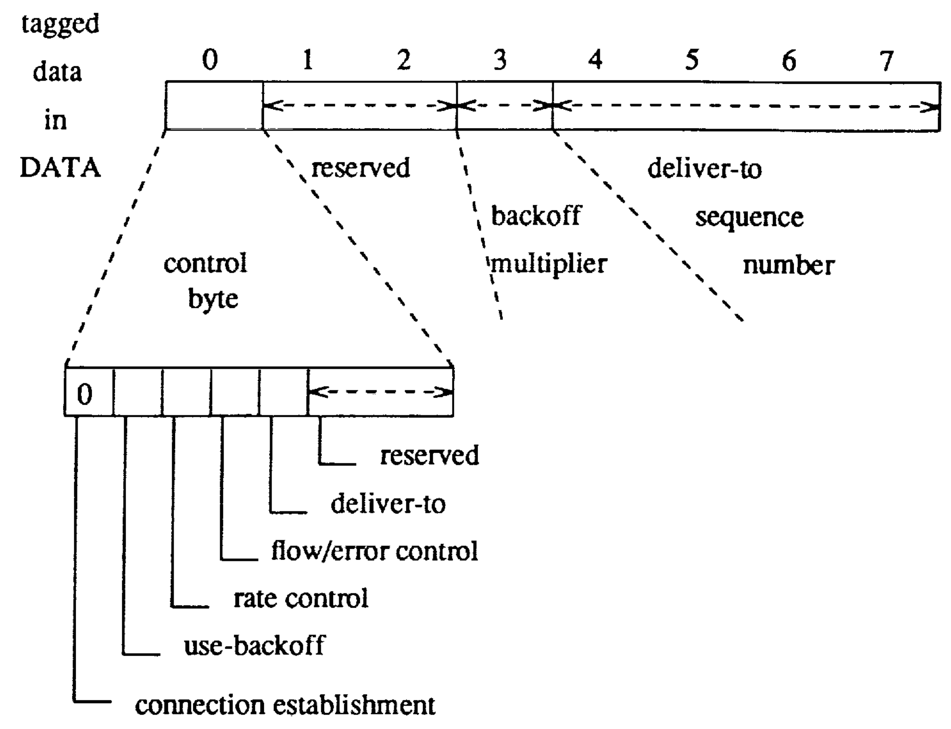
The multicast receiver opens a receive context (using `X_Mopen()`) that listens on the group address. Upon the arrival of a FIRST packet, the Multidriver checks the ETAG header bit. If ETAG is set, the Multidriver opens a transmit context (the *return context*) and sends a FIRST packet to the address to which the string in the ETAG field maps (see Figure 2).

`X_MRreply()` allows the local client process to send data to the multicast source using the return context. The parameter `xfile` provides the handle of the multicast receive context, not the return context, since the return context is managed completely internally by the Multidriver. `X_MRreply()` returns the user a special handle of type KEY for the return context. The KEY handle can be used in place of an XFILE handle in driver routines, which check for this special case. With the exception of this indirection and the loss of the tagged data feature, the return channel functions as an ordinary XTP reliable unicast connection.

The multicast receive context and its return context are always closed together and can be closed in two ways. Either the remote end transmits a close indication to the receiving context or the return context, or the local user closes the receiving context. The local user cannot close the return context directly.

3.3. Control Scheme

All tagged data in a reliable multicast transaction exchange represents driver-level control information, which is embedded in both the forward and reverse data streams. At a multicast group member, if the local application process does not generates reverse direction data, then the return context will be issuing XTP DATA packets containing only tagged data. For tagged



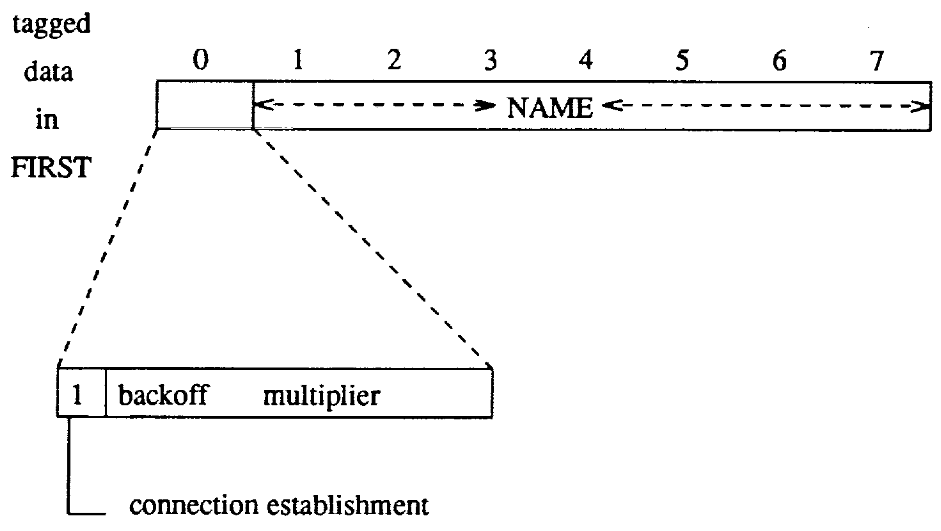


Figure 2 — Multidriver Control Scheme

data in the source-to-group (forward) direction, the first byte of the the tagged data field serves as a control byte (see Figure 2). In the reverse direction, no control byte is needed. ETAG fields carry responses to the flow/error control flag (see below), and BTAG fields carry rate control parameters.

(1) connection establishment flag — indicates the presence in the tagged data field of (1) a string that maps to the address on which to open the return context and (2) an integer denoting multicast set size. Before transmitting the FIRST packet from the return context, the Multidriver receiver waits a random amount of time between 0 and BACKOFF_TIME, which is determined by the multicast set size.

(2) flow/error control flag — requests the remote end to report one plus the sequence number of the last byte received in order at the receiving context and one plus the sequence number of the last byte that the receiving context will accept.

(3) rate control flag — requests the remote end to report its XTP rate control parameters, BURST and RATE.

(4) deliver-to flag — notifies the remote end not to deliver data to the destination process beyond the enclosed sequence number. The deliver-to flag offers the multicast source a mechanism for synchronizing message commitment.

(5) use-backoff flag — delivers an integer used for BACKOFF_TIME computation. The flag indicates that a multicast receiver must use the accompanying integer to compute the value of its BACKOFF_TIME variable and begin using a random backoff between 0 and BACKOFF_TIME for each packet transmitted from the return context.

4. Related Work

The Multidriver provides a multicast service for error, rate, and flow-controlled multidestination delivery of arbitrary-sized messages and coordinates the reliable response data channels. It does not attempt to address the message ordering and group management issues found in reliable broadcast protocols such as ([26] [4] [18]), though the Multidriver has attractive features (e.g. the deliver-to flag for atomic message delivery at all destinations) for use in building such protocols.

The transport layer multicast protocol proposed in [10] does provide for one-to-many reliable delivery and addresses the issue of many-to-one response collection. Service interfaces to the protocol allow users to select the destination group by cardinality and by explicit address. This protocol differs from the Multidriver in that the former focuses on accommodating the general case of receivers being connected across WANs as well as LANs. An experimental implementation of the protocol exists in user space under UNIX, but no performance measurements are cited.

A protocol based on Negative Acknowledgement with Periodic Polling (NAPP) ([27]) takes the novel approach of having background daemons at each receiver that assure progress and periodically send liveness messages to the source during a multicast distribution. Receivers multicast control information so that all group members overhear each other, and each control message that reaches the multicast source contains a report on all receivers' sliding windows. A primary drawback to NAPP's approach is the management of adaptive timers in the face of dynamic system parameters, changes in group size, and connections made by multicast sources of varying processing power. The defaults for the timers that drive the background daemons at each multicast group member may be inappropriate for a particular connection. Short transfers will suffer unpredictable delays and/or periods of temporary instability as timers adapt. NAPP does not provide for data in the return channels.

The Versatile Message Transaction Protocol (VMTP) ([6]) uses the transaction paradigm as the basis of all communication. It supports a multicast transaction primitive in which at least one response from the multicast group defines a successful transaction. Responses after the first one are buffered for the user and delivered if requested. Messaging service reliability depends, beyond the initial response, on the reliability of user-level transactions.

5. Performance Measurements and Functionality Demonstration

UVA XTP is designed to serve as the transfer layer component for a real-time communications subsystem such as that specified in the SAFENET standards for military and commercial ships ([12]). In the UVA implementation, XTP runs on top of a real-time, link layer messaging service ([28]). Performance measurements below were done on a single-segment Proteon ProNET-4 4 Mbps 802.5 token ring. Network nodes include ALR 25 MHz Intel 386 FlexCaches (Flexs), Zenith 16 MHz Intel 386 machines (Zeniths), a 16 MHz Intel 286 Compaq (Compaq), and 4.77 MHz Intel 8086 Leading Edge PCs (LEdges). All nodes have AT buses.

5.1. Multidestination File Transfer

Multicasting offers efficient bulk data transfer due to parallelism in delivery and message processing. Fault tolerant systems with replicated file servers, for instance, could benefit from a reliable multicast service for the delivery of file copies to the file server group. Table 1 shows the achievable throughput in delivering a large file reliably to a group of servers using the Multidriver.

Table 1

Multicast File Transfer

Transmitting Node: 4.7 MHz Leading Edge PC

Number of Receiving Nodes	*Receiving Nodes*	*Average Throughput (in Kbits/s)*	*Range*
1	Flex	174.7 Kbits/s	181.8 — 168.7
2	2 Flexs	149.1 Kbits/s	149.7 — 148.1
3	2 Flexs, 1 Zenith	112.9 Kbits/s	122.2 — 108.1
4	2 Flexs, 2 Zeniths	98.7 Kbits/s	103.3 — 97.2
5	2 Flexs, 2 Zeniths 1 Compaq	95.7 Kbits/s	100.7 — 92.7

These measurements were taken using a real-time network monitor ([24]) that timestamps each packet to an accuracy of 100 milliseconds. An application program transferred a 350,000 byte file from sender to a set of receiving nodes with one receiving context per node. The measurement began with the first packet containing data and ended when all data had been acknowledged at the sender. Error due to clock resolution is less than two percent.

Using the same driver routines, unicast transfers from the same transmitting node (a LEdge) to either a Flex, a Zenith, or a Compaq averaged 209.1 Kbits/s with a range of 200.0 - 217.0 Kbit/s. Multicasting to a group containing only one receiver causes a 16.45% drop in throughput when compared to unicasting. With two receivers, however, sequential unicasting is 29.9% slower than multicasting, and the advantage of multicasting grows with each node added. At five nodes, sequential unicasting is 56.3% slower than a multicast transfer. These figures indicate that for bulk data movement the Multidriver can achieve substantially better performance than unicast transfers and that these efficiencies are realized as soon as more than a single host has joined the multicast group.

Reliable multicast connections are necessarily bound by the slowest member of the receiving group. Hence it is not surprising that the addition of a Zenith to the receiver set of 2 Flexs produces a severe (24.3%) drop in the multicast throughput. In contrast, the addition of a second Zenith to the receiver set consisting of 2 Flexs and a Zenith or the addition of a Compaq to the set of 2 Flexs and 2 Zeniths causes much smaller drops (12.6% and 3.0%, respectively).

5.2. Implosion Control

Our testbed showed no evidence of network implosion, i.e. synchronizing receivers overrunning the transmitter's network interface. But the testbed has only a few nodes, and no more than 2 identical nodes were involved in any data exchange. Larger node populations and more homogeneous networks would be expected to have implosion problems.

However, the phenomenon of a fast node sending back-to-back packets too fast for the slow transmitter to catch the second packet was observed during early development of the Multidriver. Under some circumstances, this rate control problem caused significant delays. It appeared when timing factors resulted in a fast receiver reaching a state in which it began transmitting back-to-back packets to the transmitter. Resolution of the problem was straightforward since XTP defines rate control mechanisms. The transmitter's response contexts contain rate control parameters that are conveyed to the remote return contexts in XTP CNTL packets. The return contexts are thereby restrained to respect a minimum interpacket gap when transmitting. (The rate control flag in the Multidriver control scheme provides for rate control in the one-to-many direction.)

5.3. Multicast Transaction Latency

The Multidriver supports as a communication primitive reliable multicast transactions. Many classes of distributed applications could benefit from reliable multicast transactions. Some distributed processing entities, for instance, require a method of negotiating for access to global system resources. Examples include the locking of a global variable for updating purposes and the ability to elect, based on run-time information, particular entities out of a set to perform a necessary task ([23]). This negotiation process usually requires the exchange of short messages and may have real-time constraints as well. The dominant consideration is not, as with file transfer, greater throughput and more efficient use of network resources, but message latency. The inherent parallelism in multicasting suggests that it can produce lower latencies than a series of unicasts.

Table 2

Multicast Roundtrip Latency Transmitting Node: 25 MHz 80386 Flex 1-byte messages		
Number of Receiving Nodes	*Receiving Nodes*	*Average Roundtrip Latency (in msecs)*
1	Flex	53.2 ms
2	1 Flex, 1 Zenith	53.7 ms
3	1 Flex, 2 Zeniths	53.8 ms
4	1 Flex, 2 Zeniths 1 Compaq	54.0 ms
5	1 Flex, 2 Zeniths, 1 Compaq, 1 LEdge	58.0 ms

Table 3

Unicast Roundtrip Latency Transmitting Node: 25 MHz 80386 Flex 1-byte messages	
Receiving Node	*Average Roundtrip Latency (in msecs)*
Flex	5.63 ms
Zenith	6.85 ms
Compaq	9.63 ms
Leading Edge	17.02 ms

Table 2 and Table 3 show the measured latency experienced by the Multidriver user in sending a reliable 1-byte multicast and unicast messages from a Flex. These measurements use a clock with resolution of 50 millisecs and average over 1500 transfers.

The tables show that the Multidriver imposes a much higher latency cost than unicast, but latency costs increase slowly as the number of receiving nodes grows. For a Flex-to-Flex transfer, multicast latency is an order of magnitude greater than unicast. For the 5-node receiver set in Table 2, a series of unicast would take 45.98 ms as opposed to 58.0 ms using a Multidriver connection. By extrapolation, the experimental data suggests that the Multidriver offers lower latencies only in the case where the number of nodes in the multicast group is relatively large (e.g. generally greater than 8-10) and/or unicast latency is high with some members of the multicast group.

The relatively high minimum latency of the Multidriver scheme reflects the fact that the Multidriver control information must cross the boundary from Engine to Driver on the remote end. Unicast latencies measure Engine-to-Engine interaction since, upon reception of the FIRST packet containing the byte of user data, the receiving Engine sends back an XTP CNTL packet. The transmitting Engine receives the CNTL packet and signals the user, and the clock is stopped. In contrast, under the Multidriver, the arriving FIRST packet at the remote end contains tagged data, which must be delivered to the driver level. A FIRST packet containing the driver-level acknowledgement is then constructed and transmitted from the return context. This reverse-direction FIRST packet is acknowledged with an XTP CNTL packet at the original transmitter's response context before the user is notified and the clock stopped.

6. Conclusions

The Multidriver demonstrates that a reliable multicast transaction service can be constructed using the mechanisms provided by XTP Revision 3.4. Under this facility the user receives a reliable multicast channel that has available all the services of a unicast channel and can be accessed with the same procedural interface. Reliable response channels from every group member are available for reply collection. Mechanisms are defined for implosion control to ensure the feasibility of large multicast groups, and atomic message delivery to a group of receivers can be accommodated.

Measurements of the Multidriver facility provide strong evidence of the advantage of reliable multicast over a series of unicast for multidestination bulk data transfers. Latency measurements indicate that the Multidriver imposes a high fixed minimum latency that, in order to best multiple unicasts, must be amortized over a large (e.g. 6-10 or more) multicast group. Latency and other performance metrics would be greatly enhanced if the functionality of the Multidriver were directly supported in the underlying protocol, XTP. The principles and ideas behind the Multidriver scheme could be incorporated into XTP without much new mechanism or complexity in the protocol design. In any case, from the general view providing strong support for reliable multicasting appears necessary in the face of both escalating demand for reliable multi-party communication protocols and clear evidence of their value and feasibility, as demonstrated by the Multidriver scheme.