MTP-2: Towards Achieving the S.E.R.O. Properties for Multicast Transport

C. Bormann *J. Ott* *H.-C. Gehrcke* *T. Kerschat** *N. Seifert*

FB 3 AG Rechnernetze, Universität Bremen, D-28334 Bremen

FB Informatik, FG Kommunikations- und Betriebssysteme, Technische Universität Berlin, D-10587 Berlin

Abstract

Existing reliable multicast transport protocols have focused on achieving reliability from the point of view of senders of messages. Many applications in the area of synchronous groupware do not need this strong property, but can nonetheless benefit from a reliable multicast protocol.

For use with the *X window sharing* tool *Xy*, we have implemented the multicast transport protocol MTP as described in RFC 1301. Combined with reliable (TCP) unicast connections, MTP turns out to be a relatively good match for the transport requirements of *Xy*. Based on these experiences, we have designed a successor protocol, which aims to be more generally applicable by achieving Scalability, Efficiency, Robustness, and Ordered delivery of messages.

Keywords: Multicasting, Transport Protocol, Reliable Broadcast, Scalability, Teleconferencing.

1. Introduction

Multiparty cooperative applications have recently received much attention, as has the multicasting of datagrams in the Internet [6, 7]. The internet datagram multicasting mechanism is inherently unreliable, usually requiring a higher level protocol to achieve the level of reliability required for an application. Just as TCP has proven to be a useful basis for many applications that could in theory motivate the design of application specific transport protocols, it is likely that a generally available reliable multicast transport protocol would relieve many multiparty applications from the burden of coping with unreliable delivery in their application protocol designs.

Much of the extensive work on reliable multicast protocols has assumed relatively stable groups that need to ensure that all messages are received by all members of this well-defined group. In essence, they have focused on achieving reliability from the point of view of senders of messages: Senders were assured that messages they send either reach all members of the recipient group (possibly after removing failing members from the group) or that they receive an error indication. Recently, work on loosely coupled teleconferencing [10, 11] has directed attention to a class of multicast applications that scale up to an extent where this assumption is no longer practical. Many other applications in the area of synchronous groupware also do not need the strong property of reliability, but can nonetheless benefit from a multicast protocol providing some weaker form of reliable transport.

An interesting multicast transport protocol with a relaxed view of reliability is defined in Internet RFC 1301 [1]. For use with the *X window sharing* tool *Xy* [3], we implemented this protocol. Combined with reliable (TCP) unicast connections, MTP turned out to be a good match for the transport requirements of *Xy* [2]. Based on this experience, we have designed a successor protocol, which aims to achieve Scalability, Efficiency, Robustness, and Ordered delivery of messages.

In the next section, we give our view of the requirements for a reliable multicast protocol for teleconferencing applications. After discussing MTP in more detail, we describe MTP-2, a variant of MTP designed along these requirements, and provide a first evaluation of MTP-2.

2. The S.E.R.O. Properties — Key Requirements for Reliable Multicast Transport

There are difficult trade-offs in designing a multicast protocol. Concentrating on any single objective is unlikely to yield a generally applicable protocol. In this section, we list what we perceive to be the main requirements, in order of importance.

Scalability

While the actual usage pattern of synchronous group communication software is not yet known, it is clear that groups of wildly different sizes will need to be accommodated. A protocol that is not scalable to large groups will not be a viable multicast platform.

Many existing protocols that focus on reliability require a positive acknowledgement from each recipient to the sender of each message. This obviously does not scale to large groups. Also, group management algorithms that require an acknowledgement from each member to accept a new member are not acceptable in large groups (in particular, building a group creates an n^2 problem).

* Also Technische Universität Berlin Prozeßrechnerverbundzentrale, D-10623 Berlin.
Authors' E-mail addresses: cabo@informatik.uni-bremen.de, jo@cs.tu-berlin.de, hcg@cs.tu-berlin.de, torsten@prz.tu-berlin.de, and nilss@cs.tu-berlin.de.

Presented at the ICCCN '94, San Francisco, September 1994

This scaling problem can be circumvented by using negative acknowledgements. Unfortunately, this also conflicts with a strict reliability requirement: Not every failure will be immediately detected, since the normal behavior of a recipient, i.e. being silent, cannot be distinguished from a silent failure.

Efficiency

A reliable multicast protocol should be comparable in performance to special protocols specifically designed for an application. Just as TCP generally is slightly less efficient than a specially designed protocol would be, some more packets and additional per-packet overhead as well as some additional processing time will be tolerable. However, the protocol needs to be in the same class of overhead to be applicable to an application.

Robustness and Reliability

A reliable multicast protocol should obviously be ‘‘reliable’’ in some sense. Given the conflict with scalability, we define reliability to mean: A *recipient* can (within bounded time) find out when it is failing or being partitioned from active senders. A *sender* is assured (with sufficient probability) that all its messages reach within bounded time all recipients that are not failing or being partitioned*.

Obviously, this strict definition of reliability needs to be complemented by some measure of robustness: A protocol that declares failure in the face of trivial errors may meet this definition but is not useful. In a teleconferencing environment, a desirable robustness property is the ability to continue operating within partitions should the group become partitioned. Ultimately, the applications that use the multicast transport platform should be the ones to decide when the situation has deteriorated to a point where continuing is meaningless.

Ordering

Many applications are simplified considerably when all (or at least a certain subset of all) messages exchanged in the group arrive in the same order at all recipients, even if originated at different senders.

Obviously, any communications protocol design also needs to meet additional general requirements. In particular, a reliable multicast protocol needs to be able to adapt to different qualities of service offered by the underlying network. Also, it is useful for the reliable multicast protocol to offer unicast capabilities as a special case — otherwise, in many applications additional unicast connections will be required between pairs of members. Finally, to be successful, a protocol must be as easy to describe and implement as possible.

3. MTP as a Multicast Transport Platform

The MTP reliable multicast transport protocol is defined in Internet RFC 1301, an informational RFC [1]. MTP is designed to be used with unreliable, not necessarily sequence-preserving underlying multicast network protocols such as IP multicast. (We use the term ‘‘unreliable’’ in its technical sense of ‘‘with a loss probability that might be higher than tolerable for an application’’, not as an absolute measure of quality.)

MTP provides sequenced, globally ordered, reliable transfer of messages. It supports certain quality of service parameters, such as reliable or unreliable delivery and controlling the rate of data transfer.

3.1. The MTP Protocol

In contrast to many other transport protocols, reliability in MTP is based on negative acknowledgements (NAKs). Successful transfer of a packet is considered the normal case. Thus, MTP will benefit from the highly reliable networks being deployed.

MTP distinguishes three different roles of members of a *web* (multicast transport group): master, producer and consumer. The *master* provides the message-ordering synchronization for all members in a web. The first member of a web becomes its master. *Producers* send data in *messages* (each sent as a sequence of data packets) after obtaining a token from the designated *master*. *Consumers* receive these messages and can use negative acknowledgements to request the retransmission of packets that did not arrive.

The data transfer and retransmission is based on dividing time into *heartbeat* intervals (τ). After the initial transmission of a packet, consumers have a limited time to request retransmission of a data packet; this time is mea-

* More formally: we define the multicast transport to be reliable if for each consumer C_k that is a member of the group at the time the message M_i is submitted, exactly one of the following conditions eventually becomes valid:

i) M_i is delivered to C_k successfully, or

ii) C_k is notified that the message M_i has failed, or

iii) C_k is notified that it has missed M_i, or

iv) C_k has failed during the reception of M_i.

Messages are either delivered to a member in their entirety or not at all. The probability of messages being missed (case iii) by one or more members is defined by the QoS parameters. Graceful shutdown of a group is not considered to be in the scope of a multicast transport.

For a consumer C_k, reliability means that after it has joined the group it will receive information starting with some message $M_{s(k)}$. The consumer cannot tell in advance from which message receiving starts, nor can it influence this. When the consumer C_k finally leaves the group (intentionally, due to a termination of the group, or because of a failure) it has received a last message $M_{e(k)}$. Then the reliability property guarantees that for each message M_i with $s(k) \le i \le e(k)$, either M_i is delivered to C_k, or that C_k is notified that M_i could not be delivered (has failed or was missed).

sured in heartbeats and is referred to as the *retention* (ρ). After that time, producers are no longer obliged to honor NAKs, allowing the producer to discard its copies of the data sent; it is similar in effect to the maximum retransmission count found in many reliable data link and transport protocols.

Negative acknowledgements are sent to the producer via unicast. To know that it is necessary to send a NAK, a consumer needs to find out about the existence of a message in the first place, and about the producer responsible for this message: This is ensured by defining a minimum number of heartbeats that must be covered by a message (possibly filled up with special empty data packets)*; ρ is typically used for this number. If a consumer is unable to receive data packets for a sufficiently long time that it missed the entire message, it can find out about that by global information propagated throughout the web.

Global Ordering

The master is responsible for assigning global message sequence numbers to all messages. A producer that wants to send a message obtains a token by sending a unicast packet (`token[request]`†) to the master, which responds with a unicast packet carrying a unique sequence number (see figure 1).

Producers mark all packets of the message for which the token was granted with the message-number; they return the token implicitly with the final packet of the message (`data[eom]`). Consumers are responsible for delivering messages in the correct order to their applications.

Atomicity

A message may not reach a consumer correctly for two reasons: Either the consumer is failing/disconnected or the producer is failing/disconnected. In group communication, it is useful to have a common understanding of the group whether a message has been lost to the whole group (i.e., the producer failed) or just to a consumer.

In MTP, it is the responsibility of the master to provide this property of atomicity. It maintains a *message acceptance record*, which assigns a status to the most recent 12 messages. Messages that are in progress are marked *pending*, messages that have been correctly received by the master are marked *accepted*, and messages that have not been correctly received are marked *rejected*. The message acceptance record is transmitted in every packet sent by the master, in particular `token[confirm]` packets sent to producers. Producers

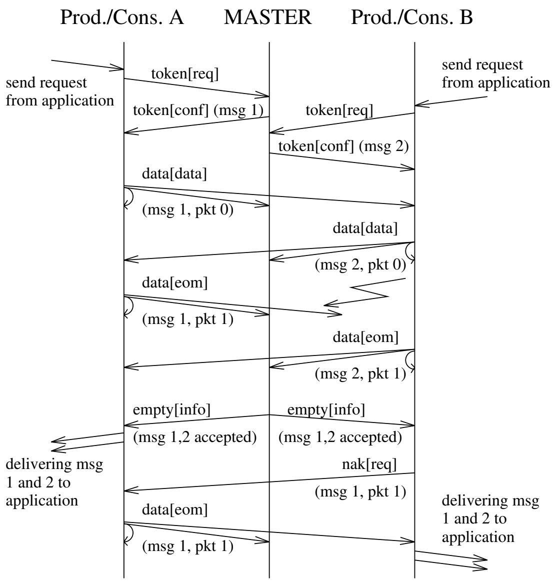

Figure 1: MTP protocol

copy the message acceptance record in each packet they send. In case no producer is active, the master multicasts `empty[hibernate]` packets, one per heartbeat.

The acknowledgement of a message by the master provides atomicity. Messages the master has not received correctly are rejected and not delivered to the application by consumers. Messages that have been accepted by masters but have been missed by a consumer are an indication that this consumer is failing (it is then within the discretion of the consumer whether this failure is reason to leave the web).

Although only two bits are needed per message, the size of the message acceptance record is necessarily rather limited as it is sent with every MTP packet. In MTP, 24 bits are set aside for this, limiting the number of concurrent messages that can be pending (in circulation) to 12.

Membership control

A process wishing to join a web sends up to ρ `join[request]` packets to a so-called "unknown" (actually, well-known) multicast address and port, supplying the "web identifier" of the web to be joined. Every master is listening to this port. Initially, a new member tries to become the master by setting the member-class flag to "master". If there is no reply, the new member becomes the master of a new web. If there is already a master running with this web identifier, `join[deny]` packets are sent to the new member, and the new member retries with a more modest membership class.

* Actually, RFC 1301 only defines a minimum number of data packets, which could in theory be sent in just two heartbeats, but we consider this to be in conflict with the basic reliability principles of this RFC.

† We follow the notation for packet types in RFC 1301, namely `type[subtype]`.

New members are only admitted when no message is pending, ensuring that no partial messages are seen by a consumer. New tokens are only granted when all pending new member applications have been processed.

Members of a web that want to leave unicast `quit[request]` packets to the master, which responds with unicast `quit[confirm]` packets. The quit request should be retransmitted ρ times, if no acknowledgement is received.

Flow Control vs. Rate Control

In most connection-oriented communication protocols, acknowledgements are used for both error recovery and flow control. MTP does not provide flow control. It does define a *window* (ν), which is a limit to the number of data packets that can be sent per active message in a heartbeat interval by one producer. This effectively limits the data rate per message (measured in packets per time unit) to $\frac{\nu}{\tau}$. In the worst case, if no other rate control policy is in effect, a consumer must therefore be able to process at a rate of $12\,\frac{\nu}{\tau}$ packets per time unit.

Reliability

Since MTP uses negative acknowledgements and producers retain old packets only for ρ heartbeat intervals, there is a nonzero probability that all retransmissions (or all NAKs) are lost and a packet never reaches a consumer. While this is not different from typical unicast protocols that have a retry limit, the real-time nature of the heartbeat intervals slightly exacerbates the problem. In particular, a temporary partitioning of duration $\rho\tau$, or simply a lost data packet followed by a temporary outage of the consumer (e.g. due to swapping) of time $\rho\tau$, causes a lost message for the affected consumer(s).

3.2. Experience

We implemented MTP on SunOS 4.1.3 with IP multicasting and UDP as underlying protocols. The protocol itself is running as a user mode process. All applications using MTP have to link to an MTP library, which starts the daemon process and opens a UNIX domain socket pair to it. A kernel mode implementation for SunOS 5.x is under way.

Our implementation of MTP nearly conforms to the specification; however, our rate control scheme limits the rate per message, not per producer.

For testing various applications, we used MTP with the recommended parameter values [1] $\tau = 200$ ms and $\nu = 20$ packets, with a message packet size of 1400 (to fit comfortably into Ethernet frames), and with a retention of $\rho = 8$. Assuming a message parallelism of 1, this limits throughput to about 140 kbytes/s, which was about the maximum we could funnel through our untuned implementation and the UNIX domain sockets.

We wrote a number of test applications, such as a multicast file transfer, a multicast version of UNIX' keyboard chat program talk(1), and a multicast audio telephone program. As can be expected from an untuned implementation, the file transfer application could not compete with TCP, but it did work well at the data rate chosen. The chat program and the telephone program were representing low delay and real-time applications; although MTP is not intended to cover real-time requirements, they held up well.

In addition, MTP is the basis for the window sharing system *Xy* [3]. On 33 MHz SPARC machines, subjectively, an X session mediated through *Xy* via MTP does not suffer any performance degradation relative to a session based on X only. Unfortunately, existing X benchmarks focus on throughput, not on responsiveness; this impression is therefore hard to substantiate. (An *Xy* mediated session generally obtains the same or better throughput than can be sustained by the X servers involved.) [2] *Xy* is a usable window sharing tool that, using a multicast transport protocol, can efficiently accommodate large numbers of participants. MTP seems to be a good match to the centralized structure implied by a window sharing tool. However, it has to be augmented by reliable unicast connections. Supporting subconferences efficiently also calls for establishing subgroups, which is rather difficult in the current IP multicast environments.

3.3. A Critique of MTP

As well as it works for the applications mentioned, MTP as defined in RFC 1301 still has some limitations. Some of these have been described in the literature [5], while others were identified in our work with *Xy*.

Master concept

The master function causes additional load on the machine on which it is being executed and its network connection, in particular membership management and token processing; every packet has to be processed by the master to update its view of the state of each producer. More significantly, the master also is a single point of failure.

Missing address management

The management of multicast group addresses is one of the less well understood issues in IP multicasting [5]. Unfortunately, the MTP specification is rather confused about various aspects of addressing; e.g., it specifies a single predefined IP multicast group to be used with all MTP multicast groups. Our MTP implementation uses the connection identifier fields of the MTP protocol as a crude mechanism to distinguish different webs during the joining phase.

Poor response to network congestion

MTP requires a bandwidth of $\frac{\nu}{\tau}$ packets per active message. If the network cannot supply that bandwidth, the protocol quickly becomes unstable. Note that early versions of TCP exhibited similar problems, which were solved by a dynamic readjustment of transport parameters [9]. In MTP, there is no well-defined way to adjust the heartbeat and retention during the connection.

Inability to process high priority traffic

In MTP, all data are of the same priority. Many applications need some form of high-priority data.

Long messages block traffic

In MTP, only a small number of consecutive messages can be active at one point in time, limited by the size of the message acceptance record. A producer that continues to send packets into a message for an extended period of time may block the entire message exchange.

No subgroups

Another requirement of some applications like *Xy*, and not covered by MTP, is the need for *subgroups*. E.g., in *Xy*, a single X client can be of interest to only a part of the conference members. It would be beneficial if the corresponding requests were relayed only to that subgroup.

No support for unicasting

Many applications including *Xy* need to send unicast data in addition to multicasting, which is not supported by MTP. *Xy*'s solution to use additional TCP connections is rather cumbersome and may cause inconsistencies in connection states between the two protocols; we think that support for unicasting should be an integral part of a multicast transport.

Flow Control vs. Rate Control

Rate control is sufficient for the normal uses that we anticipate for MTP. However, it is easily possible to overload the slowest in a heterogeneous set of recipients with a fast producer and an application that is continuously sending messages*.

Consumers can slow down the effective data rate of a sender by requesting retransmissions (the rate budget for these is taken out of the same window as for the new data transmissions); this would be a wasteful practice, however. For production use, it will be necessary to extend MTP by protocol data units that allow an overloaded consumer to reduce the data rate it receives; it will also be necessary to provide a higher level conference control mechanism with information about sluggish consumers and let it decide whether such a consumer is to be excluded from the conference.

Minimum number of packets per message

As NAKs are sent directly to the producer of a message, consumers need to receive at least one packet per message to know the producer's address. As mentioned, this is achieved by adding empty data packets to a message to guarantee a minimum of ρ packets per message. Especially when short messages predominate, this rule can become quite expensive.

* Fortunately, many such applications are concerned with the distribution of real-time data that more effectively would be distributed separately via their own multicast protocols.

4. The design of MTP-2

This section outlines the design of MTP-2, a variant of the original MTP designed to avoid some of the problems encountered in our implementation and to provide additional facilities that increase its utility for many applications.

4.1. Address Management

As mentioned, MTP has a rather confused usage of addressing. In MTP-2, an application must specify a unique *web-name* to enter a web. This name is a string (sequence of characters) of arbitrary length. The initial `join[request]` packet is sent to a special, multicast IP address that is associated with the web-name, while all other packets then use a web-specific address (to reduce processing time on the hosts) sent to the members within the `join[confirm]` packet. Addresses are chosen randomly (so they might *not* always be unique) or can be supplied by the application (which possibly has obtained it from a multicast group authority, MGA) [5]. As we can't rely on uniqueness of multicast addresses, all packets are qualified by an additional *web-id*, allowing members to distinguish data meant for different webs arriving on the same multicast IP address. This unique web-id consists of the master's IP address and its port number.

4.2. Joining Procedure

In MTP, a new member wishing to join a web has to wait until all tokens are given back to the master; this prevents new members receiving incomplete messages. This is not necessary, as the `join[confirm]` packet also contains a message sequence number. In MTP-2, new members can join a web immediately. All messages with a message sequence number older than the one given in `join[confirm]` are ignored by a new member.

4.3. Membership information for applications

In MTP-2, the master can be configured to send membership messages for each newly accepted member. Also, it can be configured to regularly check the membership status via `ismember[request]` packets.

4.4. Master loss

In MTP, the master is a single point of failure. Not only can the master fail, but a network partitioning also renders the partition without the master inoperative.

In order to achieve higher reliability, it is useful to provide a mechanism for master loss recovery. All members can detect the loss of the master when they don't receive any packets or new parameter values from the master.

The recovery procedure starts by ascertaining that the master is indeed unreachable by sending it multicast `master[suspected]` packets. If no reply is received from the master, the members agree on a new master, similar to the initial joining phase. A new web is formed using a new web-id, removing any ambiguity as to whether members still believe to be attached to the failing master.

In an additional phase, the new master accumulates information about the status of all active messages, as well as recently requested transport parameters, from all responding web members.

4.5. Master migration

Master migration is useful when the host with the running master or its network connection is overloaded, when the member to which the master is associated wants to leave or in case of an 1xN transport to move the master to the existing producer to reduce the number of `token[request]` and `token[confirm]` packets on the network. The procedure is similar to the mechanism used for master loss recovery, except that no state accumulation phase is necessary.

4.6. Dynamic adjustment of parameters

In many situations it is useful to change the parameters τ (heartbeat) and ρ (retention). When all tokens have been returned by the producers, the heartbeat can slow down to avoid needless traffic (an `empty[hibernate]` packet is sent by the master for each heartbeat).

Also, if frequent packet loss is encountered, higher values may be desired for ρ or τ, to increase the likelihood of successful (re)transmission.

Parameter changes were discussed, but no specific protocols were defined in the original MTP specification. In MTP-2, the master of a web is the only member that can set a new value for the heartbeat, retention or window parameters. Since all control packets are unacknowlegded in MTP, a method must be defined to ensure reliable propagation of a new parameter set to all consumers and producers.

If the master is actually emitting packets, after sending ρ packets, we can assume that every member has seen at least one packet*. While at least one token is granted, the master does not multicast any packets. Therefore, we introduce a new packet type `empty[info]`. It is only transmitted once in every heartbeat and only if parameters are being changed and they could not be disseminated by sending other packets. Active producers that have received such a packet copy the new parameters into their multicast data-packets. The master repeats the `empty[info]` for ρ heartbeats, unless it has seen another packet with the new parameters in the current heartbeat.

After ρ heartbeats, the new parameters have propagated throughout the web. While this propagation takes place, packets giving old and new parameters will be mixed in the web. To recognize that a packet contains new parameters, all members memorize a master state sequence number. Each time the master changes a parameter, it increments the master state sequence number. Members accept a change only if it is identified by a higher sequence number than the one stored.

Even with these mechanisms in place, there is a limit to the degree with which the master can change the parameters. If the parameters change too much in one round, a producer using a new shorter τ would destroy packets without giving a consumer still using a larger τ a chance of requesting retransmissions (the reverse scenario holds for increasing τ).

For MTP-2 we stipulate

$$\left| \tau_{new} - \tau_{old} \right| \le \frac{\tau_{old}}{\rho}$$

which is comparable in effect to using a temporary retention of $\rho - 1$.

4.7. Forwarding NAKs

To avoid the MTP requirement for a minimum number of packets per message mentioned above, MTP-2 allows consumers to send NAKs to the master or other producers if the actual producer is unknown. A producer receiving a NAK for a foreign message forwards the packet to the correct producer or to the master when it is oblivious, too. To guarantee that every consumer knows about the existence of all messages the master originates additional packets (necessary only when the web becomes quiescent).

4.8. Priority and Streams

One of the more interesting properties of MTP is its ordered delivery of messages. However, many applications do send messages that need to arrive more urgently than others.

A first step is to specify a priority in each `token[request]` packet. The master processes incoming `token[request]` packets in order of their priority. However, this doesn't solve the problem completely, because consumers can deliver high-priority messages to

* Agreement on parameter values is necessary for robustness, but not for correctness.

the application only after all other pending messages are finished, regardless of their priority.

Simply introducing a special class of high-priority messages which are not sequenced would introduce a new set of problems by inconsistent ordering between these high-priority messages. Instead, MTP-2 sequences high-priority messages as it would sequence lower priority messages.

While the MTP-2 priority controls the sequence in which message tokens are granted by the master, sequencing at the consumers is controlled by the *streams* concept: In MTP-2, every message has a priority and a related stream. Within a stream, sequencing is guaranteed, but a stream is completely independent from all other streams; therefore, messages can overtake other ones with a different stream number.

4.9. Continuing Long Messages

In MTP, only a small number of consecutive messages can be active at one point in time. A producer that continues to send packets into a message for an extended period of time may block the entire message exchange.

In MTP-2, the master marks such a message as *linked* in the message acceptance record, offering the producer to continue with the long-winded message on a message sequence number 11 higher than the original one. To inform the producer about the new message state an extra `empty[info]` packet is sent. Then the linked message can be shifted out of the message acceptance record, the offered continuation message sequence is set pending and a new token can be assigned to another producer.

The producer ends its use of the old message sequence number by qualifying one of its data packets as `data[linkeom]`

4.10. Atomicity

In MTP, consumers only deliver a message to the application when they have received acknowledgement from the master that this message was "accepted" (this is indicated in the message acceptance record propagated throughout the web). This is useful if atomicity is actually required, but causes an increase of the transmission delay usually on the order of τ, but up to the length of another currently active message.

MTP-2 defines a flag to disable the atomicity on a per message basis. Consumers are then free to hand messages to the application upon arrival of the complete packet sequence.

4.11. Client Channel

MTP's *client channel* was used as a transparent data part in every data packet, supplied by the application to partition a message into different parts. For many applications, a client channel of 8 bits is not an exact fit. Other applications do not need this feature at all. Instead of wasting more bits, MTP-2 defines a submessage-continued bit, which, when set to zero, indicates that the packet starts a new subpart of the message. Applications then can start each subpart with identifying information of the appropriate size.

4.12. Piggybacking the `token[request]` packets

In many cases a new token is required by a producer that is still sending a message. In order to reduce the number of `token[request]` packets it is useful to include a token-request flag in data packets.

4.13. Unicasting

MTP is only concerned with providing a multicast communication channel between the members. Many applications can use unicast messages, such as in collecting votes to a multicast proposal.

MTP-2 defines a separate unicast channel per producer-consumer pair. Messages sent via unicast carry the sequence number of the multicast message they are logically following, as well as a local sequence number for the unicast channel. Succeeding multicast packets (of the same or a different producer) are *not* synchronized with the unicast messages; if necessary, this can be done at the application level.

5. An Evaluation of MTP-2

In this section we provide a preliminary evaluation of MTP-2 with respect to the S.E.R.O. objectives.

Scalability

MTP and MTP-2 are both scalable up to rather large group sizes. Both protocols have little per-member overhead, particularly none after web setup if no packets are lost. The master concept seems intimidating at first, but it does not incur much more processing than for any other member of the web. No other member of the web needs to keep a membership list, if this is not necessary for the application.

MTP-2 adds robustness features important for large groups such as master migration and member information, as well as unicasting to avoid having to send all messages as multicast information.

Efficiency

A full message transaction in MTP consists of `token[request]` and `token[confirm]` unicasts, as well as at least ρ multicast `data[...]` packets, some of which may be empty. This is obviously worse than the typical unicast packet-acknowledgement pair, but much better than a data packet followed by as many acknowledgements as there are members in the group. For sufficiently reliable networks, a continuous stream of medium size MTP messages incurs very little overhead and thus is more efficient than TCP, which requires an acknowledgment for at least every other packet [4]. In MTP-2, the requirement for at least ρ multicast packets is removed, making it quite efficient for small messages, as well. To further reduce the overhead, applications could make extensive use of the submessage concept of MTP-2 to aggregate their information into larger messages*. Small numbers of packet losses can be repaired without too much loss of throughput; extremely lossy multicast distribution trees are not useful with either MTP or MTP-2, however.

MTP-2 can piggyback the `token[request]` on a `data[...]` packet in some cases. To reduce delays, MTP-2 allows the atomicity function to be switched off and some of the ordering interactions reduced by using multiple streams; also, the joining procedure is streamlined.

Robustness and Reliability

Both MTP and MTP-2 are vulnerable against rounttrip delays that exceed $\rho\tau$. MTP-2 provides a way to adjust ρ and τ dynamically. Note that with the arrival of integrated services protocols such as RSVP [12] there will be more control over delay variances, making this less of an issue.

The main contribution of MTP-2 to robustness is the ability to reform a web after the master has been lost. Also, MTP-2 copes better with multicast routing protocols that lag behind unicast connectivity.

Ordering

MTP only provides one sequence of globally ordered messages. MTP-2 allows this sequence to be split into separately progressing streams.

6. First Experience with MTP-2

MTP-2 has recently been implemented both in user mode and as a kernel driver for SunOS 5; a socket based interface is available as a library for both versions. The kernel implementation is slightly more than 10000 lines of C code, resulting in about 100 K of SPARC binary. We are now beginning performance and robustness testing. Early results that were obtained with a web of 27 stations on a live Ethernet show that our implementation is already able to transfer at about one half of the Ethernet bandwidth with a mix of SPARCstation 10/20 and SPARCclassic workstations.

* This seems to work well even with highly interactive applications such as the X window sharing tool *Xy*.

7. Conclusions

A set of requirements has been given for a generally applicable multicast transport platform. In particular, a relaxed form of reliability has been traded against improved scalability.

The RFC 1301 protocol MTP has been identified as already fulfilling many of these requirements. Meeting additional requirements led us to develop MTP-2, the salient features of which were presented.

We consider MTP-2 to be promising as the basis for a growing number of multiparty applications that will be the components of the teleconferencing applications of the future.

References

1. S. Armstrong, A. Freier, and K. Marzullo, "Multicast Transport Protocol," Internet RFC 1301, February 1992.
2. C. Bormann, J. Ott, and G. Hoffmann, "First Experience with Multicasting the X Protocol," in *Broadband Islands Third International Conference*, Elsevier, 1994.
3. C. Bormann and G. Hoffmann, "Xmc and Xy — Scalable Window Sharing and Mobility, or, From X Protocol Multiplexing to X Protocol Multicasting,," *The X Resource* **9**, January 1994.
4. R. Braden, Ed., "Requirements for Internet Hosts — Communication Layers," Internet RFC 1122, October 1989.
5. R. Braudes and S. Zabele, "Requirements for Multicast Protocols," Internet RFC 1458, May 1993.
6. S. Casner and S. Deering, "First IETF Internet Audiocast," *ACM Computer Communications Review* **22**(3), July 1992.
7. S. Deering, "Host Extensions for IP Multicasting," Internet RFC 1112, August 1989.
8. S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C. Liu, and L. Wei, "Protocol Independent Multicast (PIM): Motivation and Architecture," Work in Progress (Internet Draft draft-ietf-idmr-pim-arch-00.ps), March 1994.
9. V. Jacobson, "Congestion Avoidance and Control," in *Proceedings of SIGCOMM '88*, ACM, August 1988.
10. V. Jacobson and S. McCanne, "vat," UNIX Manual Pages, part of ftp://ftp.lbl.ee.gov/sun-vat.tar.Z, Lawrence Berkeley Laboratory, Berkeley, CA, February 1992.
11. V. Jacobson and S. McCanne, "Using the LBL Network 'whiteboard'," Presentation slides, part of ftp://ftp.lbl.ee.gov/wb/sun-wb.tar.Z, Lawrence Berkeley Laboratory, Berkeley, CA, February 1993.
12. L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, "RSVP: A New Resource ReSerVation Protocol," *IEEE Network* **7**(5), pp. 8–18, September 1993.