Abstract

Multiparty cooperative applications have recently received much attention, as has the multicasting of datagrams in the internet. The internet datagram multicasting mechanism is not reliable, often requiring a higher level protocol to achieve the level of reliability required for an application.

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. Recently, work on loosely coupled teleconferencing has directed attention to a class of multicast applications that scale up to an extent where this assumption is no longer practical.

An interesting multicast transport protocol is defined in RFC 1301. MTP provides globally ordered, receiver reliable, rate controlled and atomic transfer of messages to multiple recipients. A revised, more practical version of MTP, the Multicast Transport Protocol MTP-2 has been in use for some time.

Self-Organizing Multicast, MTP/SO, uses MTP-2 as a basis and adds spontaneous self-organization of the members of the group into local regions. Scalability is increased by providing passive group joining.
and local retransmission of lost packets.

This version of the document is not yet complete but contains most of the vital parts.

1. Introduction

Multiparty cooperative applications have recently received much attention, as has the multicasting of datagrams in the internet. The internet datagram multicasting mechanism is inherently unreliable, often 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 generally available reliable multicast protocols would relieve many multiparty applications from the details of efficiently 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 eventually received by all members of this well-defined group. Recently, work on loosely coupled teleconferencing 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 somewhat relaxed view of reliability is defined in RFC 1301 [1]. MTP can be used with unreliable and not necessarily sequence preserving underlying multicast (or broadcast) network protocols such as IP multicast. MTP provides globally ordered, receiver reliable, rate controlled and atomic transfer of messages to multiple recipients.

A revised version of MTP, the Multicast Transport Protocol MTP-2, has been used for a number of applications for some time [2]. MTP-2 has been designed to avoid some of the practical problems experienced in using MTP and introduces a number of additional facilities that increase its utility. In particular, MTP-2 no longer has a single point of failure.

This document defines Self-Organizing Multicast, MTP/SO. MTP/SO uses MTP-2 as a basis and adds spontaneous self-organization of the members of the group into a hierarchy of local regions. Scalability is increased by providing passive group joining and local retransmission of lost packets.
2. Requirements

Even more so than for unicast protocols, there are difficult trade-offs in designing a multicast protocol. It is unlikely that a single reliable multicast protocol can be applicable to all kinds of multicast applications, from a small set of replicated database systems synchronizing their updates to distributed interactive simulation systems with hundreds of thousands of processes joining and leaving large numbers of groups with high frequency.

Any design of a protocol that aims to cover a part of the ground must therefore be explicit about the specific requirements the designers had in mind. 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 that went into the design of MTP/SO, 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 with a significant rate of membership change will not be a viable multicast platform.

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

As a first level of attack, this scaling problem can be circumvented by using negative acknowledgments (NAKs). 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. There is a trade-off between scalability and the kind of reliability that can be realized.

- 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 or creates significant delays 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.

3. Overview

This section gives an overview over the protocol functions of MTP/SO. (Note to readers that have seen MTP or MTP-2: This overview is given in terms that are more generic than those used in older protocol definitions. In particular, the terms group, coordinator, sender, and receiver have been substituted for the traditional terms web, master, producer, and consumer.)

In MTP/SO there are three different roles of members in a group: coordinator, sender and receiver. The coordinator provides the message ordering for all members in a group and oversees the rate control. Senders send data in messages (each sent as a sequence of one or more data packets) after obtaining a token from the designated coordinator. Receivers receive these messages and request the retransmission of packets that did not arrive.

In MTP/SO, many actions like retransmitting control packets or requesting retransmissions depend on a time interval that is a parameter to the whole group. This interval is called heartbeat and is measured in microseconds.

3.1. Global ordering

The coordinator assigns a global sequence number to each message. In
the simplest mode of transmission, before a sender is allowed to start sending a new message, it has to obtain a token from the coordinator. This can be done by transmitting a special request packet to the coordinator or by sending the request along with data packets belonging to other messages. The coordinator answers with a confirm packet, which contains the sequence number for the new message. Senders will then send this sequence number in every data packet belonging to the message. It is the responsibility of the receivers to deliver messages in the correct order to the applications, if sequenced delivery has been specified for a message.

This results in an ordering class called "global ordering", which means that even when there are many senders simultaneously sending messages, every receiver will receive the messages in the same order which corresponds to the order in which they were sent.

As the sequencing will quite often result in an additional delay (for example when a short message is preceded by a very long one), applications can assign messages to different streams. A message is delivered irrespective of messages belonging to other streams, even if these carry lower sequence numbers. By using streams, applications can avoid unnecessary delays, simply by assigning independent messages to different streams.

A message that can be processed independent of the ones preceding it can be marked with a sequencing_off bit. Messages so marked can be immediately delivered to the application by receivers, even if the stream numbers of preceding messages are still unknown.

Normally the coordinator grants the tokens in the same order the token request packets are received. If there is a need to transmit some messages with a higher priority, applications can assign a priority to every message. This priority is only considered while granting a token (hence only when there are many tokens requested at the same time) and has no effect on the transmission rate of the message once a token has been assigned. As a result, when a sender sends messages with different priorities, it is no longer guaranteed that these are received in the same order they were queued for sending -- if they are in the same stream, they are, however, received in the same order by all receivers (including the sender).

3.2. Rate control

Rate control is overseen by the coordinator. A parameter global to the group defines the maximum throughput of the group. The coordinator dynamically adjusts a per-message parameter called window to the number of tokens granted (up to 11). Senders are not allowed to send data packets belonging to one message at an interval smaller than window (measured in microseconds). So the coordinator can ensure that the maximum throughput for the group is not exceeded.
3.3. Atomicity

At any point in time, each message is assigned a state by the coordinator: pending, accepted, or rejected.

The state of a message is set to accepted when the coordinator did receive the complete message. As soon as a sender notices one of its messages to be accepted, it sends an acknowledgment of successful transmission to its application. Such an acknowledgment does not mean that every receiver received the message. It only guarantees that at least the coordinator was able to receive it correctly. (It also provides the sequence number assigned to the message so that the application can order its own messages with respect to other messages it may have received).

A message marked as rejected was not completely received (even after requesting retransmissions) by the coordinator. Normally, every receiver will drop such a message and the sender of the message will indicate an unsuccessful-transmission error to its application.

Receivers do not deliver pending or rejected messages to the application. If a specific receiver does not completely receive a message (even after requesting retransmissions) that is finally marked by the coordinator as accepted, it will signal this as an unsuccessful-reception error to its application.

In summary, it is guaranteed that a message was either delivered correctly to every receiver, that it was delivered to no receiver and the sender is signalled an error, or that any receiver that did not receive the message is signalled an error. (Of course, the protocol works hard to minimize the number of such errors, but the above statements are guarantees of the protocol.)

Atomicity increases the message latency: applications need to wait for the accepted state propagating from the coordinator before they can act on a message. In order to allow every member to quickly learn about the state of messages, every packet contains a copy of the most recent information available about the state of the most recent messages. If application semantics do not require atomicity, unnecessary delay can be avoided by marking a message with atomicity_off.

3.4. Retransmission

Receivers request retransmissions of data packets when there is a gap in the sequence numbers of data packets received for a message or if no further data packet has arrived for more than one heartbeat while the message is still incomplete. In case all data packets for a message have been lost, this will be recognized from the message state of packets from following messages or when the coordinator
propagates the state of the most recent messages. In any case the request for retransmission can be generated at the latest after two full heartbeats.

Retransmission requests, or NAKs (negative acknowledgments) for short, are multicast to the group to reduce the implosion problem. Receivers dither the time at which they send NAKs and postpone sending a NAK when they have recently received one or more NAKs that together cover the same set of packets.

In order to answer NAKs, senders keep a copy of every data packet they sent. To limit the number of packets stored, senders are allowed to discard these copies after a defined period of time which is measured in heartbeats and depending on a special factor called retention. After retention+4 heartbeats the copies are no longer available and requests for retransmissions received after that period are denied with a special control packet. This makes sure packets are available for at least retention retransmissions.

Nonetheless there is a nonzero probability that all retransmissions (or retransmission requests) related to a packet are lost and some receivers do not receive the message correctly. For example a network partitioning that lasts longer than heartbeat*retention will result in lost messages.

This sounds undesirable, but it is similar to the retry limit used in positively acknowledged protocols, only that the normally relatively small value of heartbeat*retention puts a limit to the length of an outage that can be tolerated. We assume that the application protocol will have a way to handle receivers that experience such a long gap in reception, because it already needs a way to treat new members that appear late in the group. (Note that for applications where this is undesirable, MTP/SO could be augmented by log servers as in [3].) In any case, MTP/SO guarantees that when a message was not completely received by every receiver, either the affected receivers or the responsible sender will indicate the error to the application.

3.5. Self-organization and Repeaters

Once MTP/SO groups get large, even the handling of NAK-based retransmission traffic becomes a scalability problem. As with many scaling problems, the obvious solution is to introduce some form of hierarchy into the group. This allows at least some of the NAKs and resulting retransmissions to be handled locally within trunks and branches of that hierarchy. As MTP/SO is a many-to-many protocol, it does not make much sense to base the hierarchy on the multicast tree from any specific sender (including the coordinator, which generally is not the sole sender and which may transfer its role to another member during the activity of the group).
Instead, MTP/SO introduces the concept of a regional repeater. Receivers multicast NAKs locally before multicasting them to the entire group. Repeaters that have previously received the requested data retransmit locally after receiving a local NAK. Repeaters that don’t have the data just relay the NAK to the next higher level of hierarchy, up to the whole group (where, finally, the sender replies with another copy of the data).

A prerequisite to this mechanism is a way to do a local multicasting (of a NAK as well as of a retransmission). In current IP multicast implementations, one way to define such regions is with TTL threshold scoping; with IPv6, administrative scoping will provide a similar method. The algorithms described in the rest of this section work best when such a scoping mechanism is in effect; leaks or other imperfections in the scoping boundaries do not cause catastrophic failures, though. The following discussion assumes three levels of scopes, e.g., site, country, and continent; the exact choice of number and extent of scopes is a global parameter of the group.

With three local and one global scope, each group member is by definition in four scopes, where each local scope is contained by the next higher scope in the hierarchy. Any member that takes on a receiver role can in principle also be a repeater for any of the local scopes (each member can decide whether it wants to be a potential repeater or not, e.g. depending in the cost structure of the Internet service or on the availability of local memory space).

For scopes that contain only one member, it does not matter whether a member considers itself to be a repeater for that scope or not. For scopes that contain more than one member, a protocol is needed that makes this fact known and selects one member as the repeater. This protocol needs not necessarily ensure that there is exactly one repeater for each scope at any time, as the retransmission protocol still works without a repeater or with more than one repeater per scope, albeit less efficient.

Repeater selection should favor the ‘’best’’ member in the scope, i.e. a member that has particularly good reception from the senders, as it is most likely that this member will have received the data to be able to perform a local retransmission. Each potential repeater therefore maintains a reception quality parameter that, on a first level of approximation, tallies the quotient of the number of recently correctly received packets to the number of packets that should have been received.

Members that consider themselves repeater for a scope periodically multicast a repeater announcement message within the scope, containing the current value of the reception quality parameter. Potential repeaters observe these messages. If, within the most local scope, a potential repeater has a considerably better reception quality parameter than the current repeater, it sends a repeater announcement at the start of its next heartbeat interval and assumes the role of the repeater. Only the repeaters of the most local scope
compete for the repeater role of the next higher scope, and so on. 
(A new repeater that displaces a member that was repeater at higher 
level scopes also announces itself as repeater at these higher level 
scopes.)

To better cope with repeater failure, receivers that are not 
repeaters send NAKs at the most local scope first and escalate them 
up the hierarchy if neither a retransmission nor a more global NAK 
follows within one heartbeat. Repeaters for a set of scopes begin 
sending NAKs within the next higher scope and then escalate them the 
same way. Retransmissions always occur at the highest level of scope 
that the NAKs leading to that retransmission carried (NAKs have a 
scope field for this purpose).

A repeater that leaves a group simply sends a repeater announcement 
with reception quality zero. A repeater that crashes stops sending 
repeater announcements, causing potential repeaters to start sending 
repeater announcements after a time interval that is inversely 
related to their reception quality parameter.

3.6. Coordinator function

As it is responsible for assigning tokens and updating the message 
state, the coordinator plays a central role in MTP/SO. If the member 
carrying the coordinator function leaves the group, the coordinator 
function will be passed to one of the remaining members 
automatically.

To avoid the coordinator being a single point of failure, MTP/SO 
provides a coordinator recovery function. This allows the group to 
elect a new coordinator when the old one crashes or becomes 
unreachable. The new coordinator will then collect all information 
needed from the group members so that absolutely no information is 
lost. (This protocol should be, but is not yet, integrated with the 
repeater function.)

In order to enhance the performance of MTP/SO it may be useful to 
actively influence which member performs the coordinator function. 
For example if only one member will send messages for a longer period 
of time, the group can migrate the coordinator function to that 
member, thereby avoiding the overhead caused by requesting and 
obtaining tokens (between one and two packets for every message). 
MTP/SO allows either to request the coordinator function for oneself 
or the coordinator to pass the coordinator function to another 
member.

3.7. Membership classes

Not all members of the group will be in a position to take over the 
functions of a coordinator or of a repeater. We therefore 
distinguish several “classes” of members:
<table>
<thead>
<tr>
<th>class</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>normal member, potential coordinator and repeater</td>
</tr>
<tr>
<td>2</td>
<td>normal member, potential repeater</td>
</tr>
<tr>
<td>3</td>
<td>normal member</td>
</tr>
<tr>
<td>4</td>
<td>unreliable receiver, normal sender</td>
</tr>
<tr>
<td>5</td>
<td>unreliable member</td>
</tr>
</tbody>
</table>

Most members of an MTP/SO group will be class 1 members, i.e. they are prepared to take over the coordinator role if this is required in a coordinator recovery. Class 2 members do not want to take on this role (for application reasons or for reasons of limited resources), but compete for the repeater function. Class 3 members take over neither special function, but take part as normal members in the group; in particular, they are allowed to send NAKs.

Class 4 members never send NAKs. Their reception of messages in the group is therefore unreliable. Nonetheless, they can originate messages that are reliably received by the class 3 or higher members of the group. One way to join an MTP/SO group is to start as a class 4 member, send a message at an appropriate time, and upgrade to a higher class when the message has been accepted by the coordinator.

Class 5 members listen only; the only packet type they can send to the group is unreliable multicast datagrams (not yet described in this version of the draft). When a minimum quality of transmission/reception is defined for the group (see group[info] packets below), members may have to downgrade themselves to class 5 when they find out their own quality has dropped below the acceptable level.

4. Protocol Definition

4.1. Notational Conventions

For convenience, the datagrams transmitted by MTP/SO group members are called packets in this document.

MTP/SO packet types are written major[minor], where major is the major type of the packet and minor is the subtype within the major type. E.g., there are data[data] packets as well as data[eom] packets.

4.2. Protocol Functions and Packet Types

- Heartbeat
All members operate on a time line that is divided into heartbeats. The nominal length of a heartbeat is a global parameter of the group. The actual heartbeat boundaries (or heartbeats for short) are dithered around the nominal value. Most protocol actions are performed at the start of a new heartbeat interval. The only exception is the actual transmission of data packets, which is evenly distributed over the heartbeat interval to which the data packets are allocated.

**Global Ordering**

A sender that wants to send a message applies for a token by unicasting a token[request] packet to the coordinator. Alternatively, the sender can include a token request field in a data packet that is sent under a previously obtained token.

As soon as a token becomes available, the coordinator replies with a token[confirm] containing a new global sequence number, under consideration of the queue of token requests and the priority of the token request. The sender uses this global sequence number as the message number in every data packet pertaining to this message.

**Message Acceptance**

The coordinator maintains the message acceptance state for recent messages. For the 12 most recent messages, the message acceptance state is disseminated in every packet. Packets sent by the coordinator contain the current message acceptance state; packets sent by other members contain a copy of the most recent message acceptance state available to that sender (for data packets, this is often the state obtained via the token[confirm] packet). As the field that is used to disseminate that state only has 12 entries, the number of messages that can be pending at any point in time is limited.

To ensure that the most recent message acceptance state is always disseminated, the coordinator sends an empty[info] packet in every heartbeat in which no other member is scheduled to send packets based on tokens sent out.

**Retransmissions**

At each heartbeat, receivers that are missing packets of a message multicast nak[request] packets (see also the discussion of self-organization and repeaters above). A nak[request] contains a list of ranges of sequence numbers for one or more messages. Ranges can be open, i.e. implicitly include all further packets when the ending packet number is not known. A nak[request] that is received by a receiver postpones sending a nak[request] for the set of packets listed in the nak[request]. Empty nak[request] packets are never
sent.

If a sender no longer has a copy of the data that needs to be retransmitted, it multicasts a nak[deny] packet.

4.3. Addresses

A MTP/SO group has one group address and as many member addresses as there are members.

The member address is the combination of a 128-bit IPv6 host address (possibly in IPv4 compatibility format, i.e. with 96 bits of leading zeroes) and a 16-bit UDP port number.

The group address is the pair of a 128-bit IPv6 multicast address (again, possibly IPv4 compatible) and a group-ID. The group-ID simply is the member address of the current coordinator.

MTP/SO multicasts always use the UDP destination port number 47112 (to be assigned) and the UDP source port number from the member address. MTP/SO unicasts use UDP source and destination port numbers in the range 47112+1 to 49152-1 (note that the number 49152 marks the end of the medium priority port number space in some current IP multicast router implementations).

4.4. Packet Formats
The standard packet header contains the following fields:

- **Version**
  - For the current version of MTP/SO, version is always 3.

- **Type, Mod**
  - Packet type and type modifier (subtype).

- **Group ID**
  - For multicast packets, this field gives the member address of the current coordinator. For unicast packets, this field is not used.

- **Coordinator State Sequence Number**
  - A sequence number for the version of the coordinator state that is disseminated with this message.

- **Message Acceptance Sequence Number, Message Acceptance State Array**
  - Let \( n \) be the message acceptance sequence number, then message acceptance state array contains the most recent message acceptance states known for messages \( n-1 \) to \( n-12 \):
    - 0 pending
If the T bit is set, Number gives the serial number and Prio the priority of a token request piggybacked in this packet.

Current values for these three global parameters of the group. These parameters are given as pseudo-floating-point numbers:

<table>
<thead>
<tr>
<th>parameter</th>
<th>bits</th>
<th>mantissa (msb)</th>
<th>exponent (lsb)</th>
<th>unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>heartbeat</td>
<td>8</td>
<td>3</td>
<td>5</td>
<td>microseconds (0 to 7*2^32)</td>
</tr>
<tr>
<td>retention</td>
<td>8</td>
<td>4</td>
<td>4</td>
<td>1 (0 to 15*2^16)</td>
</tr>
<tr>
<td>window</td>
<td>16</td>
<td>11</td>
<td>5</td>
<td>microseconds (0 to 2047*2^32)</td>
</tr>
</tbody>
</table>

A token request packet is unicast from a member to the coordinator to apply for one or more tokens. Each of these requests for a token contains a serial number of that request plus a request priority. The first token request is carried in the token request part of the standard header; additional token requests can be sent in the packet type specific part following the standard header.

A token confirm is unicast from the coordinator to the member that...
requested the token. A token[cancel] can be used by the token
holding member to return the token to the coordinator.

Figure 4: data packets (except data[eom])

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        stream number          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|A|0 0 0|O|  L | Message Sequence Number |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       Packet Sequence Number |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Data                                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The S bit, if set, indicates that ordered delivery is not required
for this message ("sequencing_off"). The A bit, if set, indicates
that atomic delivery is not required for this message
("atomicity_off"). The O bit ("original") is set only for the
first transmission of the data packet by the original sender. It is
reset for any kind of retransmission (regardless whether performed by
the original sender or not). L ("level") is a binary number
ranging from 0 to 3. Level 0 indicates a global transmission; levels
1 to 3 indicate transmission of the packets at the second most global
to most local level scope, resp. (For a retransmission, the
transmission level indicates the scope in which this data packet was
sent; lower level repeaters can use this information to decide
whether they can defer their own retransmissions.)
To ensure that the original sender of a message becomes known even if the only packets a receiver has received from this message were repeater retransmissions, the data[eom] packet differs from the other data packets in that it contains a copy of the original sender’s member address. (Note that this information is redundant for packets that have the O-bit set; it is retained in favor of a common packet format for all cases.) With an optional authentication protocol (not specified in this version of the document), authentication information can be given with this last packet of the message; the length in 32-bit words is in AL.
### Figure 5: nak[request], nak[deny]

<table>
<thead>
<tr>
<th>0</th>
<th>Message Sequence Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Packet Sequence Number (Low)</td>
</tr>
<tr>
<td>0</td>
<td>Packet Sequence Number (High)</td>
</tr>
</tbody>
</table>

The F bit, if set, indicates that, starting at the packet sequence number (low), all packets from the given message are missing. As with data packets, L gives the scope level at which this NAK is being multicast/replied to. NAK request and deny packets inhibit the transmission of further such packets from other potential transmitters (for one heartbeat) only at the level of scope given. A retransmission that is a response to a NAK request should be sent at the level of scope given.

### Figure 6: status[request], status[deny]

<table>
<thead>
<tr>
<th>0</th>
<th>Message Sequence Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Packet Sequence Number (Low)</td>
</tr>
<tr>
<td>0</td>
<td>Packet Sequence Number (High)</td>
</tr>
</tbody>
</table>

A status request packet can be multicast by a member to request status for messages that already have scrolled off the message acceptance state array in the standard header. A status deny response indicates that the retention time for keeping information about the status of the messages has passed.
Responding to status requests, a repeater (for local scopes) or the coordinator can multicast status info. The U bit, if set, indicates that the status of the given message is unknown. The S field gives the message acceptance state as in the message acceptance state array.

The K-bit, if set, indicates that reliable receiver status (membership class 1 to 3) is intended, i.e., that an explicit acknowledgment for this member has to be given within a group[info]. The C-bit, if set, indicates that the transmitter is a potential coordinator (membership class 1); it causes other potential coordinators with a higher member address to back off. The scope field gives the actual scope in which this packet was transmitted (this cannot just be given as a scope level number as the actual scope levels used in this group may not yet be known to the transmitter).
The group[info] packet is periodically transmitted by the coordinator and by each repeater to ensure that all group members are aware of the global parameters of the group and of the quality of the current repeater.

Two parameters give dynamic information about the transmitter and about the group: Quality is the (0,16 bit fixed point) product of reception and transmission quality of the transmitter. Activity is a measure for the recent activity of this group (useful for merging decisions by applications).

The other fields of the packet give global group parameters that usually are constant: The U-Bit (‘unreliable’), when set, indicates that this group operates entirely without NAKs and retransmissions. The E-Bit (‘elect’) is set for group[info] packets originated by the coordinator in case it is willing to transfer the coordinator function to a higher quality member; it requests other potential coordinators to announce their quality (if better) via group[info].

L gives the scope level, and, indirectly, the source of the group[info]: level 0 packets are originated by the coordinator or by other potential coordinators (the latter if the source address is not equal to the coordinator part of the group address), level 1 to 3 packets are originated by repeaters of the respective level.

Analogously, the TTL fields provide the TTL scopes of the levels: TTL 0 is the scope of the entire group, TTL 1 to TTL 3 give the scopes of
the most global to most local repeater levels. Setting the scope for a level to zero indicates that this level is not in use. The fields minimal send quality and minimal receive quality give minimum levels of quality for a member that wants to send reliable messages or that wants to request retransmissions (reliable reception); if not met, they cause the member to assume a lower membership class.

At the end of the fixed part of group[info] packets, extensions can be added. Their type is identified by a one-byte type code their length given by a one-byte length field, giving the number of 32-bit words beyond the initial one in this extension.

Figure 9a: group[info] extension for member acks

```
+------------------------------------------+
|  1 |  4 | (Port part) |
+------------------------------------------+
|------------------------------------------+
| (Address part) |
+-------------------+
| Acknowledged |
+-------------+
| Member-Address |
+-------------+
+-------------------+
```

Type 1 group[info] extensions are used to carry an acknowledgment for a group[seek] requests by a member that needs to achieve reliable reception status quickly (K-bit in group[seek] set).

4.5. Summary of packet types

<table>
<thead>
<tr>
<th>packet type</th>
<th>type[code]</th>
<th>multi/uni</th>
<th>sent by</th>
<th>see Figure</th>
</tr>
</thead>
<tbody>
<tr>
<td>data[data]</td>
<td>0[0]</td>
<td>m</td>
<td>C,R,s</td>
<td>4</td>
</tr>
<tr>
<td>data[eom]</td>
<td>0[1]</td>
<td>m</td>
<td>C,R,s</td>
<td>4a</td>
</tr>
<tr>
<td>data[daily]</td>
<td>0[2]</td>
<td>m</td>
<td>C,R,s</td>
<td>4*</td>
</tr>
<tr>
<td>data[ceom]</td>
<td>0[3]</td>
<td>m</td>
<td>C,R,s</td>
<td>4*</td>
</tr>
<tr>
<td>nak[request]</td>
<td>1[0]</td>
<td>m</td>
<td>r</td>
<td>5</td>
</tr>
<tr>
<td>nak[deny]</td>
<td>1[1]</td>
<td>m</td>
<td>C,s</td>
<td>5</td>
</tr>
<tr>
<td>group[info]</td>
<td>2[0]</td>
<td>m</td>
<td>C,R</td>
<td>9</td>
</tr>
<tr>
<td>group[seek]</td>
<td>2[1]</td>
<td>m</td>
<td>C,R,s,r</td>
<td>8</td>
</tr>
<tr>
<td>quit[order]</td>
<td>3[0]</td>
<td>u</td>
<td>C,R</td>
<td>*</td>
</tr>
<tr>
<td>token[request]</td>
<td>4[0]</td>
<td>u</td>
<td>s</td>
<td>2</td>
</tr>
<tr>
<td>token[confirm]</td>
<td>4[1]</td>
<td>u</td>
<td>C</td>
<td>3</td>
</tr>
<tr>
<td>status[request]</td>
<td>5[0]</td>
<td>m</td>
<td>C,R,s,r</td>
<td>6</td>
</tr>
<tr>
<td>status[deny]</td>
<td>5[1]</td>
<td>m</td>
<td>C</td>
<td>6</td>
</tr>
</tbody>
</table>

Bormann, Ott, Seifert
coord[suspected]  6[0]  m  R,s,r  *)
coord[established]  6[1]  m  C  *)
coord[seek]  6[2]  m  C  *)

multi/uni: m is multicast, u is unicast.

sent by: C is coordinator, R is repeater, s is sender, r is receiver.

*) Not yet described in the present version of the document.

5. References


6. Authors’ addresses

Carsten Bormann
Universitaet Bremen FB3 TZI
Postfach 330440
D-28334 Bremen, GERMANY
cabo@informatik.uni-bremen.de
phone +49.421.218-7024

Nils Seifert, Joerg Ott
Technische Universitaet Berlin FR6-3
Franklinstrasse 28/29
D-10587 Berlin, GERMANY
nils, jo@cs.tu-berlin.de
phone +49.30.314-73389