Supporting Multiple Quality of Service Options with High Performance Groupware*

Gregory V. Chockler Nabil Huleihel Idit Keidar
Danny Dolev

Email: {grishac,nabil,idish,dolev}@cs.huji.ac.il
Url: http://www.cs.huji.ac.il/{~grishac,~nabil,~idish,~dolev}

Institute of Computer Science
The Hebrew University of Jerusalem
Jerusalem, Israel

Technical Report CS96-3

March 1996

*This work was supported by Ministry of Science of Israel, grant number 032-7658

Abstract

Reliability carries different meanings for different applications. For example, in a replicated database setting, reliability means that messages are never lost, and that messages arrive in the same order at all sites. In order to guarantee this reliability property, it is acceptable to sacrifice real-time message delivery: some messages may be greatly delayed, and at certain periods message transmission may even be blocked. While this is perfectly acceptable behavior for a reliable database application, this behavior is intolerable for a reliable video server. For a continuous MPEG video player [18, 17], reliability means real-time message delivery, at a certain bandwidth; It is acceptable for some messages to be lost, as long as the available bandwidth complies with certain predetermined stochastic assumptions. Introducing database style reliability (*i.e.* message recovery and order constraints) may violate these assumptions, rendering the MPEG decoding algorithm incorrect.

A desktop and multi-media conferencing tool [16], is a *Computer Supported Co-operative Work* (CSCW) application incorporating various activities such as video transmission and management of replicated work space. These activities obviously require different qualities of service, and yet are part of the same application. Furthermore, CSCW applications often need to be fault-tolerant, and need to support smooth reconfiguration when parties join or leave. Groupware [1] is a powerful tool for the construction of fault-tolerant applications, providing reliable multicast and membership services with strong semantics. In this paper, we incorporate multiple quality of service options within the framework of groupware systems. This way, a single application can exploit multiple quality service options, and can also benefit from the groupware semantics.

1 Introduction

In this paper we present a novel communication paradigm that provides multiple *quality of service (QoS)* options for multi-media and *Computer Supported Cooperative Work (CSCW)* [16] applications, *e.g.* video conferencing. We incorporate different *QoS multicast channels* within the groupware framework; the channels are an extension of *group communication systems (GCSs)* that provide reliable multicast services with strong order constraints as well as group membership services. A channel multicasts a segment of messages with a specific QoS option, independently of messages that are not part of the segment. Each message segment is transmitted with the QoS guarantees requested for this segment, as fast as this QoS allows. Stronger order semantics are preserved for the entire segment and not for single messages in it. This allows us to selectively impose order guarantees relative to a subset of the messages, independently of other messages. As a test case, we describe the implementation of QoS channels in two GCSs: Transis and Horus [1].

Our approach is most suitable for applications that require high bandwidth fast multicast, with different QoS requirements for different messages, *e.g.* where most of the multicast has weak order and reliability constraints, but for a small portion of "critical" messages reliability and strong order constraints are vital. The strong semantics and the reliable ordered multicast provided for the "critical" messages can make the service fault-tolerant, without slowing down the mass of the messages. GCS membership services allow the system to smoothly reconfigure when parties join or leave.

Typical applications that may benefit from the concept presented in this paper include video conferencing, replicated video on demand servers, cooperative network management, and many more. Figure 1 depicts a video conferencing application, with a new party wishing to join the discussion. In Section 6 we describe several examples of applications that can benefit from our services among which is a multi-media and desktop conferencing application [16].

1.1 The Benefits of Groupware

Groupware is a powerful tool for the development of fault-tolerant distributed applications and CSCW services: GCSs provide the application builder with reliable multicast services with several message ordering paradigms, and with group membership services that guarantee strong semantics. Some of the leading GCSs today are: Transis, Totem, ISIS, Horus, Psync, Relacs, RMP, and Newtop; A survey of GCSs may be found in [1].

A GCS usually runs in an environment in which processes and communication links can fail, and in which messages may be lost or arbitrarily delayed. In such "inconvenient" environments, the GCS simulates to its application a "benign" world in which message delivery is reliable within the set of reachable (live and connected)

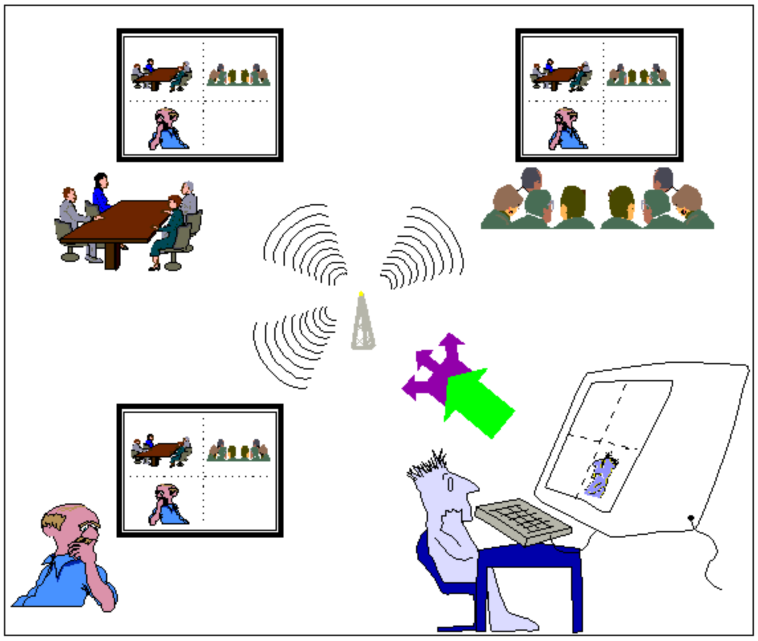


Figure 1: A New Party Joining a Video Conference

processes. The GCS also reports to the application which processes are reachable at any given time. For example, the *Virtual Synchrony* [7], *Strong Virtual Synchrony* [9] and *Extended Virtual Synchrony* [14] models provide powerful semantics, that greatly facilitate application design [7, 3, 10, 5]. GCSs today have begun to exploit new technologies, and to run over fast networks *e.g.* ATM [6] in WAN environments.

1.2 Currently, Groupware is Problematic

In spite of its strengths, groupware is rarely used for applications that utilize high throughput fast multicast, and require reliability for only a small portion of their multicasts. Many multimedia applications are in this category, *e.g.* MPEG video transmission consists of a few full images, which must be reliably delivered, followed by incremental update frames which can be sent unreliably [17]. The loss of these incremental frames will not seriously damage the video image. Such applications often exploit fast networks, *e.g.* ATM, and do not recover lost packets (unless they result in the loss of full image frames).

Such applications do not use GCSs because they require a different QoS than provided by the GCS. Strong semantics, order and reliability guarantees significantly slow down the application which is adequately supported by a "mostly reliable" multicast. Most GCSs provide only reliable multicast services. Reliability requires message buffering, managing acknowledgments for messages and retransmissions; Thus message delivery is delayed, especially when messages are lost. Order constraints further increase the delay. Consequently, smaller bandwidth is available. This can negatively affect the application if, for example, the ATM constant bit rate QoS is requested. Furthermore, the notion of reliability for an MPEG application is ironically different than the GCS notion of reliability: MPEG decoding algorithms make assumptions regarding stochastic network bandwidth variability. The introduced delays may violate these assumptions, rendering the decoding algorithm incorrect.

Some GCSs provide unordered and unreliable multicast services. For example, the Horus system allows an application to send unreliable messages. However, within the same connection to Horus, only one semantics is supported, so in this case all the messages have to be unreliable. Applications can open more than one connection and send different messages in different connections to get different semantics, but then no interconnection semantics is provided. It is possible to extend Horus so that more than one semantics will be supported in the same connection*. A similar approach was taken in other systems, *e.g.* Highways [2], where messages with weak order constraints interleave with messages with higher constraints. However, such an approach is inadequate for the applications mentioned above.

This approach suffers from one main drawback: There's no possibility to selectively enforce order guarantees relative to a subset of the messages, independently of other messages. Thus, two unrelated transmission channels (*e.g.* multicasting two separate video channels or two large files) interfere with each other. Moreover, an unreliable message is either delayed by all messages with stronger ordering constraints, or, alternatively, is delivered independently of all other messages. Often, neither solution is appropriate: *e.g.* in an MPEG application, incremental update frames must follow the corresponding full picture frame only, and have no restrictions w.r.t. other messages. In a typical multi-media application, several types of data streams (*e.g.* video, audio, translation subtitles) are each multicast independently [18], as shown in Figure 2, and are synchronized at the receiving server.

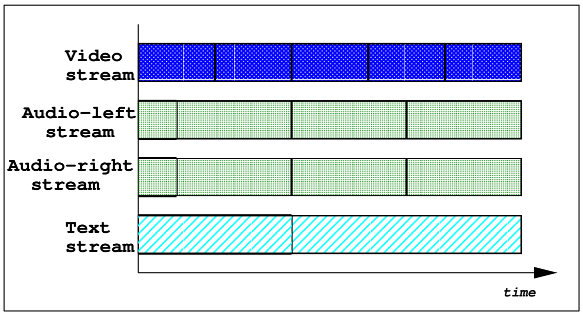


Figure 2: Data Streams in a Multi-media Application

*This idea was suggested by the Horus group.

1.3 QoS Multicast Channels within Groupware

We propose to incorporate several QoS channels into a groupware framework. A QoS channel is a segment of multicast messages from a single source. A channel can support any of several QoS options: A channel may directly provide the underlying network properties, *e.g.* the constant bit rate QoS of ATM. Alternatively, it may recover messages to provide reliable multicast. A channel may be unordered or may enforce FIFO order. The messages within a channel neither interfere with messages from other channels, nor with other messages multicast via the GCS. Thus, messages within a channel may only be delayed because of other messages in the same channel, due to the reliability or order constraint of the particular channel.

Each message segment is "wrapped" with a shell that is multicast via the reliable services of the GCS, and thus semantics are provided for the entire segment, as if it were a single message (or two messages: *begin* and *end*). This way the application may benefit from the powerful semantics of membership service, and order guarantees may be enforced among channels and regular reliable messages. The messages within the segment are transmitted independently of the regular message flow in the GCS.

The challenge was to provide these semantics without imposing a high overhead on messages within channels, so that the overall performance will not be degraded because of the support for other QoS options of the GCS. Messages are delayed within a channel *only* due to an application request to order the entire message segment relative to other messages. In other words, the system imposes the minimum delay needed to enforce only the order constraints *explicitly* requested by the application.

The integration between the reliable multicast and QoS channels has the following advantages:

- A single application can exploit assorted QoS options.
- Applications can be made fault tolerant using powerful group communication semantics (such as virtual synchrony). These semantics hold for the channel as a whole, not for specific messages within a channel. This is usually what the application requires.
- The entire channel is ordered with respect to reliable messages and other channels. This feature is useful for CSCW applications.
- Reliable "critical" messages may be used for synchronization and checkpointing, *e.g.* full image messages in MPEG.

2 The Environment and Model

A set of processes communicate over the network. The system is asynchronous: there is no bound on relative process speeds or message delay.

We consider the following types of failures: The network may partition into several components†, and remerge. Processes may crash and recover. A message may be lost by all members of a component, or only by part of them. The network may duplicate messages, and it provides no message sequencing guarantees. We assume that failures are detected using a (possibly unreliable) fault detector, *e.g.* a timeout mechanism.

2.1 Multicast Services

In this paper we discuss the implementation of several types of multicast services. The basic operation in a multicast service, is to post a message to a set of processes. A process may send a message to any subset of the processes. The multicast service *delivers* the message to its multiple targets. Multicast services are characterized by two properties: reliability and order constraints.

Reliability A multicast service is *reliable* if it delivers each message exactly once to each of its currently operational and connected targets‡, overcoming message omission and duplication. Otherwise the service is *unreliable*.

Order constraints There exist various levels of constraints on the order of message delivery. Typical examples of order constraints are:

- **None** – no order constraints.
- **FIFO** messages from a single source are delivered in the order of their transmission.
- **Causal** messages are delivered in an order preserving the "happened before" (causal) partial order defined by Lamport [12]. The causal order is defined as the transitive closure of: $m \xrightarrow{cause} m'$ if $\text{deliver}_q(m) \rightarrow \text{send}_q(m')$ or if $\text{send}_q(m) \rightarrow \text{send}_q(m')$.
- **Agreed** messages are delivered in the same order at all targets. This order preserves the causal partial order.

Order constraints cause a delay in message delivery: *e.g.* if a process sends two reliable FIFO messages m_1 and m_2, and m_1 is lost, the delivery of m_2 will be delayed until the recovery of m_1. Moreover, messages may be further delayed by stronger order constraints of preceding messages.

† A **component** is sometimes called a **partition**. In our terminology, a partition splits the network into several components.

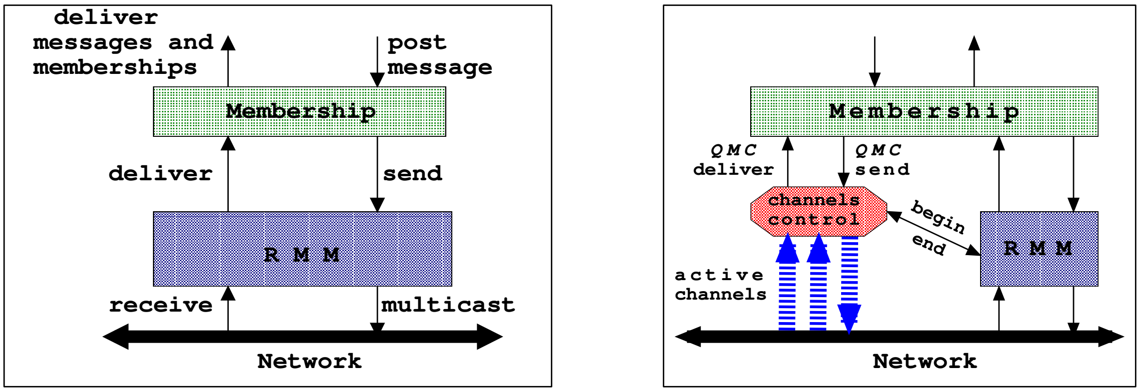
‡ Reliable multicast is sometimes defined to guarantee delivery at all *correct* targets. This definition is not appropriate for systems that tolerate network partitions, where two processes may be disconnected, and yet both are correct. In this paper we require delivery at connected targets only. This reliability guarantee is sometimes called *atomic*.

2.2 Group Multicast and Membership

Group communication systems provide several levels of reliable multicast services, as well as group membership services. A *group communication system (GCS)* supports reliable multicast communication among *groups of processes*. The basic communication primitive is to post (*send*) a message to a group. The GCS multicasts the message over the network to other instances of the GCS. The instances of the GCS receive the message from the network, and *deliver* the message to all the members of the group.

After a group is created, the group undergoes *membership changes* when new members are added to the group and when members are taken out of the group. The membership service of the GCS reports these changes to the application through special *membership change* messages, that contain a unique *membership identifier* and a list of connected processes. Membership change messages are delivered among the stream of *regular* messages. Thus, during the execution of an application, the GCS delivers to it a sequence of regular messages interposed by membership change messages.

The task of the GCS is to simulate to the application an environment in which message delivery is reliable within the set of reachable (live and connected) processes, and give the application an indication which processes are reachable at any given time. The *Virtual Synchrony* [7], *Strong Virtual Synchrony* [9] and *Extended Virtual Synchrony* [14] models provide powerful semantics. For example, they guarantee that if two processes p and q deliver the same two consecutive membership changes M_1, M_2, then for every message m that p delivers between M_1 and M_2, q also delivers m between M_1 and M_2.



(a) Basic Groupware (b) Groupware with QMC

Figure 3: Groupware Structure

A GCS may be seen as composed of two modules (or layers): a *reliable multicast module (RMM)* and a membership module, as depicted in Figure 3(a). The RMM delivers messages through the membership module, which delivers messages and membership changes to the application. The membership module reports to the RMM when membership changes occur. A GCS delays message delivery in order to guarantee the reliability and order constraints.

3 Quality of Service Multicast Channels (QMC)

We propose to incorporate *quality of service multicast channels (QMC)* into group communication systems (GCSs) that provide reliable multicast. The reliable multicast module (RMM) of the GCS is extended with quality of service multicast channels with weaker reliability and order constraints.

A quality of service (QoS) channel multicasts a segment of messages from a single source, independent of messages which are not multicast via the channel. The messages that are multicast via a QoS channel neither interleave nor interfere with the regular flow of messages in the RMM or in other channels, and therefore are not delayed by them. The whole segment is ordered relative to reliable messages and other channels.

A message segment is "wrapped" with two reliable messages: *begin-channel* and *end-channel* that are multicast via the RMM. The application builder chooses which of the various reliable multicast services provided by the GCS RMM to use for these messages, in order to guarantee the order constraints required for his application. The structure of groupware with QMC is depicted in Figure 3(b).

Notation: For channel C we denote by M_C the set of messages sent via the channel C, by S_C the sender of M_C. *begin-channel*$_C$ and *end-channel*$_C$ are the corresponding *begin-channel* and *end-channel* messages.

3.1 QoS Multicast Channels Types

A QMC is characterized by the following parameters: the ordering level of its shell (*begin-channel* and *end-channel* messages) and the constraints on reliability and order of the messages within the channel. All the messages belonging to the same channel are delivered before the *begin-channel* message and after the *end-channel* message, and all have the same constraints. A channel is labeled according to the constraints on messages within the channel. We propose the following types for a QoS channel C:

Unreliable Unordered The unreliable unordered QMC preserves the properties of the underlying network. The implementation: A message $m \in M_C$ is delivered immediately upon its reception from the network.

Unreliable FIFO For any $m_1, m_2 \in M_C$ such that m_1 was sent before m_2: if both messages are delivered then m_1 is delivered before m_2. The implementation: A message $m \in M_C$ is delivered immediately upon its reception from the network if no successor of m in M_C was already delivered. Otherwise m is discarded.

Reliable Unordered All the messages in M_C are guaranteed to be delivered exactly once. No ordering constraints are guaranteed for any message in M_C. The implementation: A message $m \in M_C$ is delivered immediately upon its reception from the network provided that m was not delivered previously. Lost messages are recovered using negative acknowledgments and retransmissions.

Reliable FIFO All the messages in M_C are guaranteed to be delivered exactly once and the FIFO delivery order is preserved. The implementation: A message $m \in M_C$ is delivered if all of m's predecessors in M_C have been delivered. If m was previously delivered it is discarded, and otherwise m is buffered.

3.2 Extensions

In this section we propose possible extensions to QMC. The first extension deals with **prioritized channels**. This service is based on the cyclic-UDP [20] protocol, which is a best-effort priority-driven network protocol especially designed for video transmission [11]. The prioritized channel is *unreliable*, but it does use retransmissions in order to increase the probability of successful delivery. The messages are multicast via the channel sorted according to their priority: the first multicast message has the highest priority, and the last message – the lowest. The probability of successful delivery of a message is proportional to its priority.

The implementation of the prioritized channel is similar to that of [20]: The *prioritized* channel C retransmits messages as long as *end-channel*$_C$ was not delivered. Messages from M_C are stored in a queue, and are multicast in bursts of a fixed size. Each target sends a list of negative acknowledgments (NACKs) for each burst, requesting retransmission of lost messages. For each burst, the sender scans the queue from the beginning, so that higher priority NACKed messages are retransmitted first in the burst. This way, higher priority messages get more chances for retransmission, and therefore have a better chance to be delivered. Furthermore, these messages are multicast earlier within the burst, and therefore an underlying unreliable multicast protocol (*e.g.* UDP) has a better chance to deliver them.

The second extension deals with **nesting of channels**. Unreliable and prioritized channels may be nested within reliable channels: the *begin-channel* and *end-channel* of the inner channel are delivered as reliable messages within the outer channel, and not via the RMM. This allows order constraints to be enforced among channels without introducing redundant constraints, and thus increases the liberty to selectively choose order dependencies.

For example, an MPEG video server multicasts picture frames, each followed by unreliable incremental update frames. The picture frames delivery must be FIFO and reliable, while the increment frames may be unreliable, and are ordered w.r.t. the preceding picture frame. Picture frames can be multicast via a reliable FIFO channel, and increments – in nested unreliable channels. This implementation imposes the order relationship between pictures and increments, and imposes no order constraints w.r.t. other messages or channels. Video representation languages (*e.g.* Rivl [19]) can be naturally extended to "compile" into this form of representation (for video transmission).

3.3 QMC in Presence of Membership Changes

The membership messages of the GCS have an important role in providing strong semantics such as virtual synchrony. These semantics restrict the order of membership messages w.r.t. regular messages. QMC provides these semantics for the entire channel, and not for particular messages within the channel. This is usually what the application requires.

Whenever a membership change occurs the following situations are possible for channel C:

- Some process p disconnects from S_C before $\textit{end-channel}_C$ is received. In this case, p delivers a special $\textit{broken-channel}_C$ notification, indicating that some messages in M_C are unrecoverably lost. The $\textit{broken-channel}_C$ message automatically forces C to be closed.
- Some process p proceeds together with S_C. In this case, S_C continues to multicast and p continues to deliver C's messages.
- Some new process p joins S_C. In this case, p delivers a special $\textit{join-channel}_C$ notification, and following it delivers the remainder of M_C.

4 QMC Implementation

We now discuss the implementation of QoS multicast channels (QMC) within the GCS framework. The implementation makes use of three services: an unreliable multicast facility (*e.g.* UDP), a reliable multicast module (RMM) and the membership service of the GCS.

4.1 QMC Application Interface

The QMC application interface consists of three functions: *open* a QoS channel, *multicast* a message via an open QoS channel, and *close* an open channel. For a channel C, the *open* function multicasts $\textit{begin-channel}_C$ along with C's type, and the *close* function multicasts $\textit{end-channel}_C$. Both messages are multicast via the RMM using one of its reliable multicast services, or, if the channel is nested, via an outer reliable channel. The type of service is determined according to the application's request. Prototypes of these functions are described in Figure 4.

- *QoS-Open(channel-type, message-type, user-data)*
  Multicasts, via the RMM, a message of type *message-type* containing *user-data* and an indication that this message opens a QoS channel of the type *channel-type*. Returns the unique *channel-id* that is used for further reference to the channel. This message is called *begin-channel*.

  For a nested channel, the *message-type* is replaced by the *channel-id* of the outer channel.

- *QoS-send(channel-id, user-data)*:
  Multicasts a message containing *user-data* through the open QoS channel *channel-id*.

- *QoS-Close(channel-id, message-type, user-data)*
  Multicasts, via the RMM, a message of type *message-type* containing *user-data* and an indication that this message closes the QoS channel. This message is called *end-channel*.

  For a nested channel, the *message-type* is replaced by the *channel-id* of the outer channel.

Figure 4: The QMC Application Interface

4.2 Data Structures

Each channel has a single sending process, where the channel was opened. The number of channels that can be simultaneously open for sending by the same process is bounded, the process maintains a list of its open send-channels. A channel C is identified through a unique channel identifier Id_C. Each $m \in M_C$ is identified through a pair $Id_m = <Id_C$, counter$>$. The QMC *multicast* function stamps each message m with Id_m and multicasts m through the unreliable multicast facility (bypassing the RMM).

Each process also maintains a list of QoS channels open for delivery (active channels): *active-list*. Whenever a *begin-channel*$_C$ (*end-channel*$_C$) is delivered, Id_C is inserted into (removed from) *active-list*. A record in *active-list*, corresponding to a channel C, contains channel's type: $C.type$, and the maximum counter value of messages previously delivered via this channel: $C.MaxDelivered$[§].

4.3 Unreliable QMC Message Delivery

Upon reception of a message $m \in M_C$ from the network, Id_C is looked up in *active-list*. If Id_C is in *active-list*, then the following situations are possible: First, if C is an unordered channel, m is immediately delivered. Second, if C is a FIFO channel and

[§] $MaxDelivered$ is relevant only for unreliable FIFO channels.

```
HANDLE-MESSAGE-DELIVERY(m)
{
    let C be m's channel id.
    if (C ∉ active-list) then
            discard m;
            return;
    fi
    if (C.type is FIFO) then
        if (m's counter <= C.MaxDelivered) then
            discard m;
            return;
        fi
        C.MaxDelivered = m's counter;
    fi
    deliver m to the user;
}
```

Figure 5: Message Delivery in Unreliable QMC

m's counter is greater than $C.MaxDelivered$, m is delivered and $C.MaxDelivered$ is set to m's counter. If not, m is discarded. If Id_C is not in *active-list* then m is discarded[¶]. Pseudo-code describing the message delivery is presented in Figure 5.

4.4 Handling Membership changes

When a membership change message is delivered, each source multicasts the list of its open channels. Open channels with newly joined senders are joined: For each such channel C, *join-channel*$_C$ is delivered to the application and Id_C is inserted into *active-list*. Open channels with senders that have crashed or detached, are dropped: For each channel C in *active-list* such that S_C is not a member of the new membership, C's record in *active-list* is removed and *broken-channel*$_C$ is delivered to the application.

4.5 Reliable QMC Implementation

The implementation of the reliable QMC is a bit more complicated: lost messages have to be recovered. This requires buffering of messages for retransmission. A garbage collection protocol discards buffered messages when their reception is acknowledged by all targets. The delivery of the *end-channel* message is delayed until all the channel messages are delivered. In the case of reliable FIFO channels the delivery of the messages that arrive out of order is additionally delayed.

[¶]As an optimization the messages that arrive before *begin-channel*$_C$ can be buffered.

Buffering and retransmissions introduce some difficulties that are solved by means of a flow control (FC) mechanism. In particular, the following two problems are addressed: buffers overflow and the network overload. The role of FC is to slow down the transmission rate if there are many messages the reception of which is not acknowledged by all the members of the current membership. A detailed description of FC implementation using the *network sliding window* approach can be found in [4].

5 QMC Incorporation in Transis and Horus

We now present the incorporation of QMC in the Transis and in the Horus [1] GCSs. QMC could be similarly implemented in other GCSs.

5.1 Incorporation of QMC in Transis

Transis provides three types of multicast services: Causal, Agreed[‖] and *Safe*. The Safe messages are ordered in Agreed order w.r.t. all other messages of any type and are guaranteed to be delivered by all the members of current membership unless they crash.

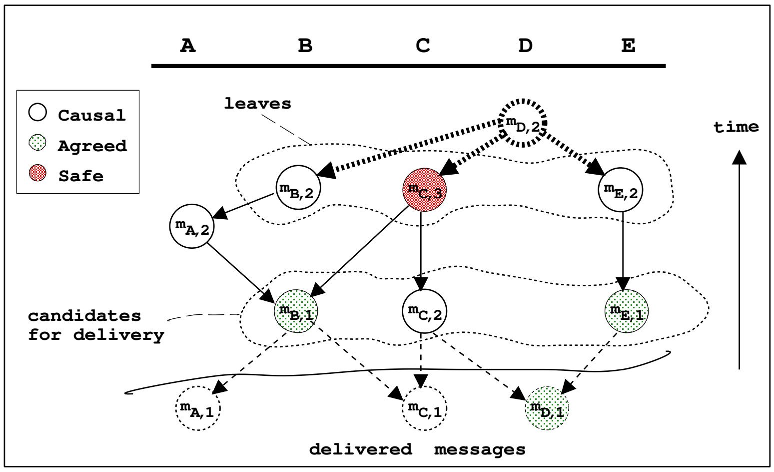


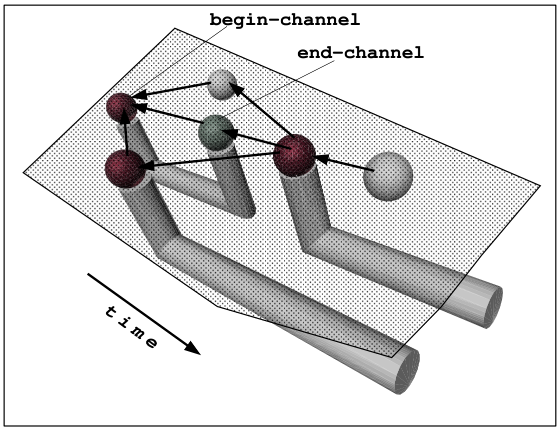
Figure 6: The Transis DAG

The main data structure of the Transis reliable multicast module (RMM) is a directed acyclic graph, *DAG*, (see Figure 6) based on Trans [13] and on Psync [15]. The DAG contains only messages that have no missing causal predecessors. Each new message emitted by a process p contains piggybacked acknowledgments (ACKs) for all the messages that p received and inserted to the DAG so far; it is enough to ACK only messages that are leaves in the DAG. There is an arc from a message m' to a message m in the DAG if m' contains an ACK to m. The ACKs are used by the

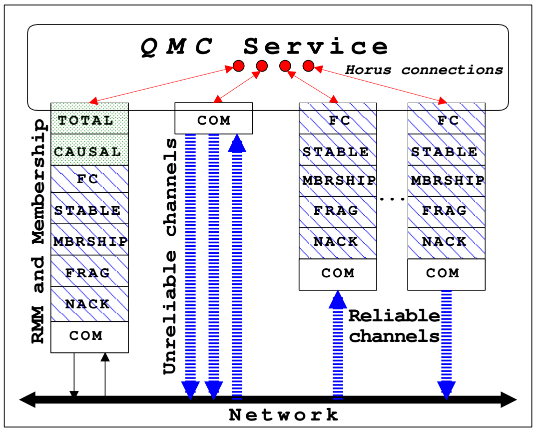
[‖]The Causal and Agreed services are defined in Section 2.1.

target processes in order to detect lost messages and reconstruct the causal relations between messages (for details see [8, 4]). Notice that the DAG reflects the causal relationship among messages.

Since all Transis services preserve the causal order, a message does not become deliverable until it causally follows only delivered messages. All the deliverable messages in the DAG form the *set of candidates*. The messages that were sent via the Causal multicast service are delivered as soon as they become candidates. The Agreed or Safe messages may be further delayed in the set of candidates because they demand a higher level of coordination among the processes before delivery. Transis employs sophisticated algorithms for the Agreed and Safe delivery. These algorithms are based solely on the DAG structure without exchange of additional messages [8].



(a) The QMC 3-D DAG in Transis



(b) QMC in Horus

Figure 7: QMC Incorporation in Groupware

We extend the DAG data structure of the Transis RMM to incorporate QoS multicast channels. The *begin-channel* and *end-channel* messages are multicast using the DAG. The regular channel's messages are passed "in the background" and do not intervene with the regular flow of messages in the DAG. Intuitively, this mechanism resembles a three-dimensional DAG, as depicted in Figure 7(a). The channel's messages are delivered after the corresponding *begin-channel* message is delivered and before the *end-channel* message is delivered.

5.2 Incorporation of QMC in Horus

The Horus system has a layered structure. Each type of multicast service is implemented as a separate layer, stacked on top of weaker service layers. QMC may be easily incorporated in Horus. The service opens several connections to Horus as depicted in Figure 7(b):

- One reliable connection (that represents the RMM) stacked on top of various reliable service layers and a membership layer. QMC multicasts regular messages (that are not part of any channel) via this connection. This connection is also used for *begin-channel* and *end-channel* messages, and for getting an indication of the membership. This guarantees virtual synchrony semantics of entire channels w.r.t. other channels, regular messages, and membership changes.

- One unreliable unordered connection (bypassing all reliable layers), for unreliable channels.

- A separate reliable connection for each reliable channel.

The channel service needs only take care of ordering channel messages w.r.t. the corresponding *begin-channel* and *end-channel* messages, as described in Section 4.

6 Applications and Concluding Remarks

In this paper, we presented the notion of quality of service multicast channels (QMC) in group communication systems (GCSs). Below we describe a few applications that may exploit QoS multicast channels within GCSs.

Multi-media and desktop conferencing systems are described in the survey of CSCW systems [16]. Such a system consists of several conferees (users), that cooperatively use a variety of application such as a meeting room (video and audio), shared work space (*e.g.* cooperative editing or drawing on a board), etc. The *conference agent* controls the communication among the conferees and the applications. We now describe how a distributed agent can exploit GCS with QMC to provide the services listed in [16]:

Multicast Application Inputs and Outputs: A typical application is a meeting room which requires video and audio multicast. Video and audio are multicast independently in separate channels [18]. Prioritized channels are most suitable for video transmission [11, 20]. Audio transmission uses an unreliable FIFO channel.

Other applications (*e.g.* cooperative editing) manage shared data, that has to be consistently replicated. GCS semantics provide a powerful tool for replication, as demonstrated in [10, 5, 3].

Floor Control: The agent allows input for a certain application to come from one conferee, who is currently authorized to provide input. This is analogous to passing the pen for the board in a meeting. GCS semantics is easily used to guarantee that exactly one conferee holds the pen, and to give conferees a fair chance of getting the pen.

Work Space Management: The agent consistently replicates the layout, grouping and placement of shared windows among all conferees. Conferees may alter the layout, and the changes will be identically reflected on all the conferees' screens. As mentioned above, replication is easily supported using GCS.

Dynamic Reconfiguration: Conferees may dynamically join and leave meeting rooms. Each addition or departure reconfigures the system. The GCS membership service provides an indication when conferees join and leave. The ordering semantics of channels w.r.t. membership changes allow for smooth and consistent reconfiguration at all remaining members, with no need for additional communication.

Logging: The agent is often requested to record the conference. All the communication in the system is routed through the agent, and thus the agent can easily record it.

Replicated video on demand servers, may exploit QoS channels to transmit video. The servers can consistently share information while each is serving clients. If one server crashes or detaches from a client, the other servers get an indication, and can smoothly take over. This is similar to the approach taken in the implementation of [11] in Horus.

Our experience in [3] shows that a *distributed system management application* can greatly benefit from the semantics of GCS. *Distributed software installation and upgrade* application uses GCS to group target machines with the same installation requirements into a *single* multicast group. Software packages are multicast to the group by means of a reliable FIFO channel. The GCS membership service allows to recognize hosts on which installation could be incomplete because of a network partition or host crash.

The performance of QoS multicast greatly depends on the size of the segments, and on the mixture of different QoS options, as well as on the properties of the underlying network. There is a tradeoff in determining the ideal segment size: if segments are small the overhead of handling *begin-channel* and *end-channel* messages is high. On the other hand, large segments are less fault tolerant: failures may cause loss of synchronization, and applications re-synchronize at the end of segments. The longer the segment, the longer it takes to re-synchronize. It is our hope that further work on this topic will identify segment sizes that induce low overhead and yet provide reasonable quality of service for specific applications.

Acknowledgments

We are thankful to Ken Birman for his helpful comments and suggestions. Special thanks to Tal Anker, David Breitgand, Zohar Levy and the other members of the Transis project for their valuable remarks.

References

[1] *Communications of the ACM 39(4), special issue on Group Communications Systems*, April 1996. To appear.

[2] M. Ahuja. Assertionas about Past and Future in Highways: Global Flush Broadcast and Flush-vector-time. *Information Processing Letters*, 48(1):21–28, October 1993.

[3] Y. Amir, D. Breitgand, G. Chockler, and D. Dolev. Group Communication as an Infrastructure for Distributed System Management. In *International Workshop on Services in Distributed and Networked Environment*, number 3rd, June 1996. To appear.

[4] Y. Amir, D. Dolev, S. Kramer, and D. Malki. Transis: A Communication Sub-System for High Availability. In *FTCS conference*, number 22, July 1992.

[5] Y. Amir, D. Dolev, P. M. Melliar-Smith, and L. E. Moser. Robust and Efficient Replication using Group Communication. Technical Report CS94-20, Institute of Computer Science, The Hebrew University of Jerusalem, Jerusalem, Israel, 1994.

[6] T. Anker, D. Breitgand, D. Dolev, and Z. Levy. The WAN according to GARP. In preparation.

[7] K. Birman and T. Joseph. Exploiting Virtual Synchrony in Distributed Systems. In *Symp. Operating Systems Principles*, number 11, pages 123–138. ACM, Nov 87.

[8] G. Chockler, N. Huleihel, and D. Dolev. ARTOP: An Adaptive Randomized Total Ordering Protocol. In preparation.

[9] R. Friedman and R. V. Renesse. Strong and Weak Virtual Synchrony in Horus. TR 95-1537, dept. of Computer Science, Cornell University, August 1995.

[10] I. Keidar and D. Dolev. Efficient Message Ordering in Dynamic Networks. In *ACM Symp. on Prin. of Distributed Computing (PODC)*, 1996. To appear.

[11] D. Kozen, Y. Minsky, and B. Smith. Efficient Algorithms for Optimal Video Transmission. TR 95-1517, Computer Science Department, Cornell University, May 1995.

[12] L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7):558–565, July 78.

[13] P. M. Melliar-Smith, L. E. Moser, and V. Agrawala. Broadcast Protocols for Distributed Systems. *IEEE Trans. Parallel & Distributed Syst.*, (1), Jan 1990.

[14] L. E. Moser, Y. Amir, P. M. Melliar-Smith, and D. A. Agarwal. Extended Virtual Synchrony. In *International Conference on Distributed Computing Systems*, number 14th, June 1994.

[15] L. L. Peterson, N. C. Buchholz, and R. D. Schlichting. Preserving and Using Context Information in Interprocess Communication. *ACM Trans. Comput. Syst.*, 7(3):217–246, August 89.

[16] T. Rodden. A survey of CSCW systems. *Interacting with Computers*, 3(3):319–353, 1991.

[17] L. A. Rowe, K. D. Patel, B. C. Smith, and K. Liu. MPEG Video in Software: Representation, Transmission, and Playback. In *High Speed Networking and Multimedia Computing, IS&T/SPIE Symp. on Elec. Imaging Sci. & Tech.*, February 1994.

[18] L. A. Rowe and B. C. Smith. A Continous Media Player. In *Int. Workshop on Network and OS Support for Digital Audio and Video*, number 3, November 1992.

[19] B. C. Smith. RIVL: A Resolution Independant Video Language. Submitted for publication. Available in http://www.cs.cornell.edu/Info/Faculty/bsmith/rvl-tcl.ps.

[20] B. C. Smith. *Implementation Techniques for Continous Media Systems and Applications*. PhD thesis, University of California at Berkeley, 1994.