Multimedia Multicast Transport Service for Groupware*

Chockler, Gregory V., Huleihel, Nabil, Keidar, Idit, and Dolev, Danny,
The Hebrew University of Jerusalem, Jerusalem, Israel†

1.0 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 [20, 19], 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. Many CSCW groupware and multimedia applications require *quality of service* multicast for most of their messages, and may greatly benefit from reliable multicast for a small portion of "critical" messages. Furthermore, such applications often need to be fault-tolerant, and need to support smooth reconfiguration when parties join or leave. Group communication [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 Multimedia Multicast Transport Service that supports multiple quality of service options within the framework of group communications systems. This way, a single application can exploit multiple quality service options, and can also benefit from the group communication semantics.

2.0 Introduction

In this paper we present a novel communication paradigm that provides multiple *quality of service (QoS)* options for multimedia and *Computer Supported Cooperative Work (CSCW)* [18] applications, *e.g.* video conferencing. The *Multimedia Multicast Transport Service (MMTS)* consists of multiple QoS options incorporated in the group communication framework. The MMTS extends *group communication systems (GCSs)* that provide reliable multicast services

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

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

with strong order constraints as well as group membership services. The MMTS allows to multicast a bunch of messages with a specific QoS option, independently of messages that are not part of the bunch. Each message bunch is transmitted with the QoS guarantees requested for this bunch, as fast as this QoS allows. Stronger order semantics are preserved for the entire bunch 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.

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. For example, *negotiation and re-negotiation* of QoS [14, 13] is a major challenge in the design of video multicast applications; it requires all the communicating parties to agree on certain QoS parameters, adjusted to actual capabilities of the communicating parties. The strong semantics and the reliable ordered multicast provided by the GCS for the "critical" messages provides a simple and efficient solution for QoS negotiation. Furthermore, it 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.

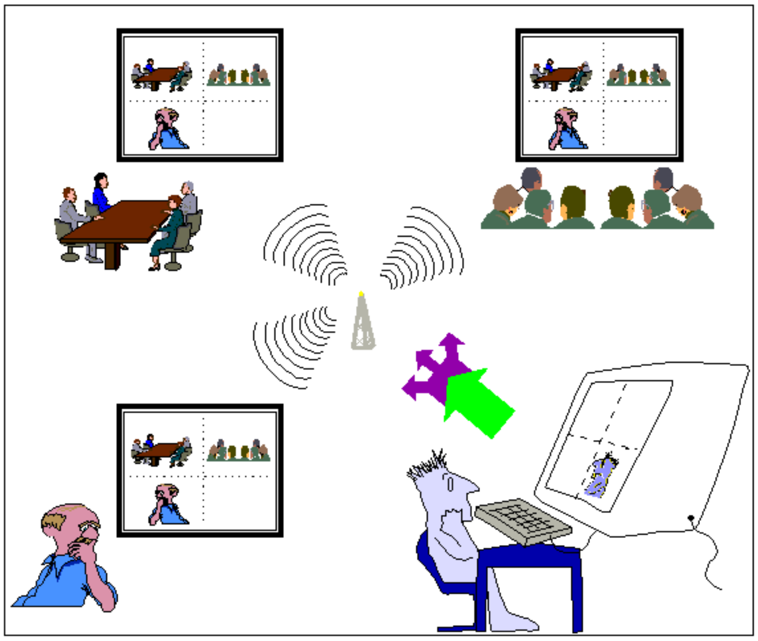


Fig. 1 A New Party Joining a Video Conference

Typical applications that may benefit from the concept presented in this paper include video conferencing, video multicast (pay TV), replicated video on demand servers, cooperative network management, and many more. **Fig. 1** depicts a video conferencing application, with a new party wishing to join the discussion. In Section 6.0 we describe several examples of applications that can benefit from our services among which is a multimedia and desktop conferencing application [18].

2.1 The Benefits of Group Communication

Group communication systems (GCSs) are powerful tools for the development of fault-tolerant distributed applications and for CSCW groupware and multimedia applications. GCSs provide the application builder with reliable multicast services with several useful message ordering paradigms, *e.g.* totally ordered[1] multicast greatly facilitates coordination and QoS negotiation in video applications. GCSs also provide group membership services that guarantee strong semantics, which are useful for the construction of fault tolerant applications. 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].

GCSs introduce the notion of the *group abstraction* that allows the application builder to treat multiple communicating parties as a single connection. Members may dynamically join and leave the group (*i.e.* subscribe and unsubscribe to the connection). The group abstraction allows for smooth reconfiguration of discussion groups: a process that sends messages to the group does not need to keep track of the changes in its membership. A multicast group is identified by the *logical name* assigned to it when the group is created. Each message targeted to the group's logical name is guaranteed to be delivered to all the currently connected and operational group's members. The group abstraction may also be exploited for reducing the transmission rate for slow receivers by multicasting the video concurrently to different groups for receivers with different QoS capabilities [7]. Receivers can dynamically switch from group to group, according to the bandwidth they have available. Another approach to reducing the bandwidth for slow receivers is based on *filtering* [14]: the video frames are multicast to several target groups, one group receives a self-contained low bandwidth (and hence low quality) video film, all the receivers subscribe to this group. Increment frames that improve the video quality are multicast to the other groups. Receivers with higher processing capabilities subscribe to one or more these groups, and thus improve the quality of the video they receive, according to their capabilities.

Other multimedia transport services, *e.g.* ACCOPI [13] also use group abstraction for multimedia multicast, but they usually support only $1 \longrightarrow N$ logical topologies (one sender, many receivers). This is justified because complex connections in the transport service complicate the design of the transport protocol. However, using a GCS arbitrary connection topologies are simple and efficient to implement. This allows to provide each application with the topology most suitable for its needs.

2.2 Currently, GCS is Problematic

In spite of their strengths, GCSs are 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. For example, in video multicast applications, reliability and order semantics are required for coordination and for QoS negotiation messages, but not for the video transmission.

Such applications do not use GCSs because they require QoS that is not 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

[1] Totally ordered multicast is also called atomic multicast.

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 a video application is ironically different than the GCS notion of reliability: video 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 that preserve the underlying network properties. 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, with a GCS, a 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 a typical multimedia application, several types of data streams (*e.g.* video, audio, translation subtitles) are each multicast independently [20], as shown in **Fig. 2**, and are synchronized at the receiving server. Ordering constraints must be preserved within each stream, but messages from different streams should not interfere with each other. Another example is video transmission: an MPEG-encoded video consists of a few full images which must be reliably delivered [19], followed by incremental update frames which can be sent unreliably. The incremental update frames must follow the corresponding full picture frame only, and have no restrictions w.r.t. other messages.

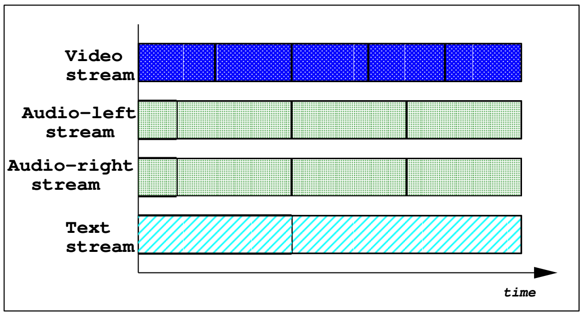


Fig. 2 Data Streams in a Multimedia Application

2.3 Multimedia Multicast Transport Service within Group Communication Systems

We incorporate a *Multimedia Multicast Transport Service (MMTS)* that provides several QoS options into the group communication framework. The basic object in MMTS is a *bunch*. A bunch is a *container* object that consists of messages. All the messages in the bunch are multicast by the same source, have the same multicast group of targets and obey to the same QoS requirements assigned to the bunch. A bunch may directly provide the underlying network properties, *e.g.* the constant bit rate QoS of ATM. Alternatively, messages within a bunch may be recovered to provide reliable multicast. A bunch may be unordered or FIFO order may be enforced. The messages within a bunch neither interfere with messages from other bunches, nor with other messages multicast via the GCS.

The GCS serves as a *coordination* object for coordinating different instances of the bunch object. The coordination object (GCS) interacts only with the bunch object as a whole and does not interact with individual bunch messages. Each bunch is "wrapped" with a shell that is multicast via the reliable services of the GCS, and thus semantics are provided for the entire bunch, 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 bunches and regular reliable messages. The messages within the bunches 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 bunches, so that the overall performance will not be degraded because of the support for coordination. Messages are *only* delayed within a bunch either due to an application request to order the entire message bunch relative to other messages or because of other messages in the same bunch, due to the reliability or order constraint of the particular bunch's QoS. 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 group communication and MMTS the following advantages:

- A single application can exploit assorted QoS options.
- Applications can be made fault tolerant using powerful group communication semantics. These semantics hold for the bunch as a whole, not for specific messages within a bunch. This is usually what the application requires.
- The entire bunch is ordered with respect to reliable messages and other bunches. This feature is useful for CSCW applications.
- The GCS provides support for dynamic reconfiguration. This allows cooperating parties to freely join/leave multicast groups and therefore offers a good abstraction for CSCW groupware applications.
- Reliable "critical" messages may be used for coordination and checkpointing, *e.g.* QoS negotiation and re-negotiation.

The object-oriented design of our system has the following advantages: First, an implementation of a specific QoS is encapsulated in the corresponding bunch object and hidden from the coordination object. This allows the coordination object to manipulate different bunch objects in the generic way according to the application defined coordination policy. Second, the application builder can easily combine different QoS options to form the QoS mixture desirable for the application. Finally, new QoS options can be smoothly integrated into the system.

3.0 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: Processes may crash and recover. The network may partition into several components[2], and remerge. 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] A **component** is sometimes called a **partition**. In our terminology, a partition splits the network into several components.

3.1 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[3], 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: $m \overset{cause}{\longrightarrow} m'$ if $\mathrm{deliver}_q(m) \rightarrow \mathrm{send}_q(m')$ or if $\mathrm{send}_q(m) \rightarrow \mathrm{send}_q(m')$.

Totally Ordered (Atomic) 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.

3.2 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

[3] 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*.

application an indication which processes are reachable at any given time. The *Virtual Synchrony* [6], *Strong Virtual Synchrony* [9] and *Extended Virtual Synchrony* [16] models provide powerful semantics, that greatly facilitate application design [6, 2, 10, 3]. 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.

GCSs today have begun to exploit new technologies, and to run over fast networks *e.g.* ATM [5] in WAN environments.

4.0 Multimedia Multicast Transport Service (MMTS)

We propose to incorporate *Multimedia Multicast Transport Service (MMTS)* into group communication systems (GCSs) that provide reliable multicast. The GCS is extended with Multimedia Multicast Transport Service providing QoS multicast.

The basic MMTS object is a *bunch*. A bunch is a *container* object that consists of messages. The messages within a specific MMTS bunch neither interleave nor interfere with the regular flow of messages in the GCS or in other bunches, and therefore are not delayed by them. The whole bunch is ordered relative to reliable messages and other bunches. The application builder defines the inter-bunch coordination policy in terms of the general bunch object. Each bunch object implementing a specific QoS type inherits from this object.

A message bunch is "wrapped" with two reliable messages: *begin-bunch* and *end-bunch* that are multicast via the GCS. The application builder chooses which of the various reliable multicast services provided by the GCS to use for these messages, in order to guarantee the order constraints required for his application. The structure of GCS with MMTS is depicted in **Fig. 3**.

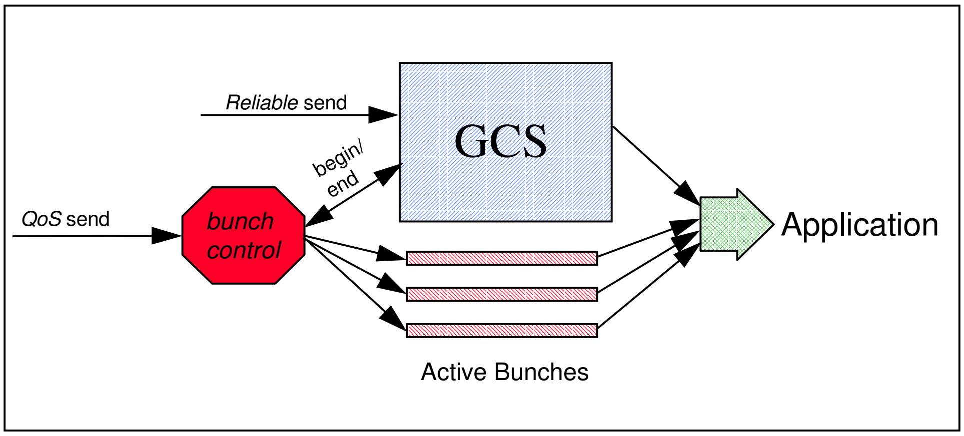


Fig. 3 The MMTS Structure

Notation: For bunch B we denote by M_B the set of messages sent via the bunch B, by S_B the sender of M_B. *begin-bunch*$_B$ and *end-bunch*$_B$ are the corresponding *begin-bunch* and *end-bunch* messages.

4.1 QoS Multicast Bunches Types and Structures

A QoS bunch is characterized by the following parameters: the ordering level of its shell (*begin-bunch* and *end-bunch* messages) and the QoS requirements of the messages within the bunch. Messages belonging to the same bunch are delivered before the *end-bunch* message and after the *begin-bunch* message, and all have the same QoS constraints. A bunch is labeled according to the constraints on messages within the bunch. The following are examples of QoS types for a bunch B:

Unreliable Unordered The unreliable unordered QoS bunch preserves the properties of the underlying network. the network.

Unreliable FIFO For any $m_1, m_2 \in M_B$ such that m_1 was sent before m_2: if both messages are delivered then m_1 is delivered before m_2.

Reliable Unordered All the messages in M_B are guaranteed to be delivered exactly once. No ordering constraints are guaranteed for any message in M_B.

Reliable FIFO All the messages in M_B are guaranteed to be delivered exactly once and the FIFO delivery order is preserved.

Prioritized Bunch implements a more complex QoS type, especially designed for video transmission [11]. This service is based on the cyclic-UDP [22] protocol, which is a best-effort priority-driven network protocol. The prioritized bunch is *unreliable*, but it does use re-transmissions in order to increase the probability of successful delivery. The messages within the bunch are 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 MMTS allows the application writer not only to easily incorporate new QoS types into the system but also to alter the bunch's structure in order to create more sophisticated QoS types. For instance, interesting QoS types can be created if **nesting of bunches** is supported. Unreliable and prioritized bunches may be nested within reliable bunches: the *begin-bunch* and *end-bunch* of the inner bunch are delivered as reliable messages within the outer bunch, and not via the GCS (see **Fig. 4**). This allows order constraints to be enforced among bunches without introducing redundant constraints, and thus increases the liberty to selectively choose order dependencies.

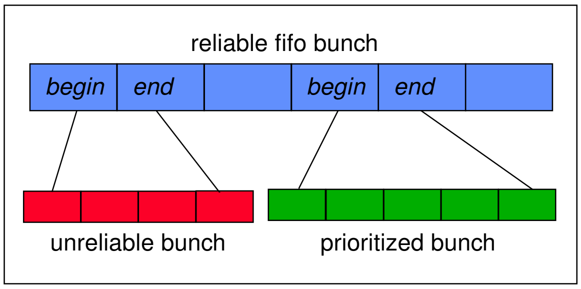


Fig. 4 Nesting of Bunches

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 bunch, and increments – in nested unreliable bunches. This implementation imposes the order relationship between pictures and increments, and imposes no order constraints w.r.t. other messages or bunches. Video representation languages (*e.g.* Rivl [21]) can be naturally extended to "compile" into this form of representation, for video transmission.

4.2 MMTS 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. MMTS provides these semantics for the entire bunch, and not for particular messages within the bunch. This is usually what the application requires.

Whenever a membership change occurs the following situations are possible for bunch B:

- Some process p disconnects from S_B before *end-bunch*$_B$ is received. In this case, p delivers a special *broken-bunch*$_B$ notification, indicating that some messages in M_B are unrecoverably lost. The *broken-bunch*$_B$ message automatically forces B to be closed.
- Some process p proceeds together with S_B. In this case, S_B continues to multicast and p continues to deliver B's messages.
- Some new process p joins S_B. In this case, p delivers a special *join-bunch*$_B$ notification, and following it delivers the remainder of M_B.

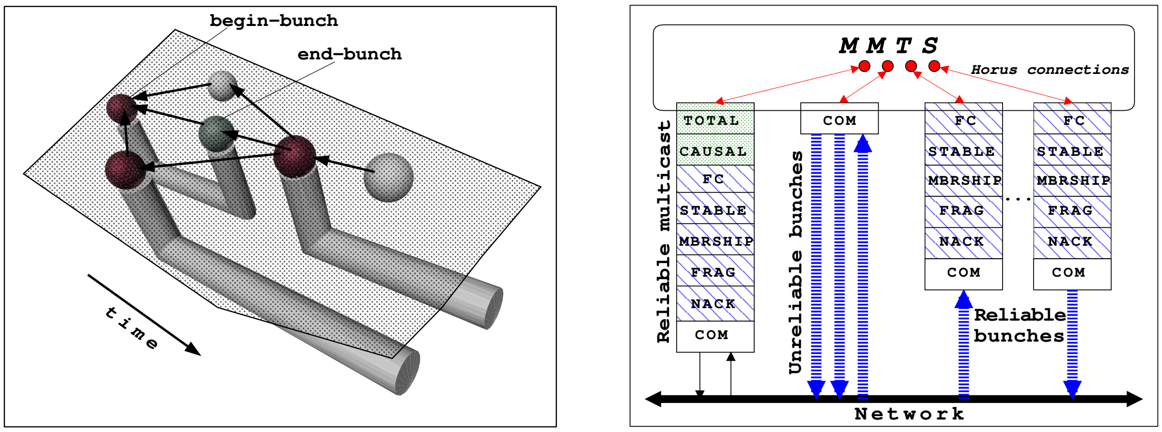
5.0 MMTS Incorporation in Transis and Horus

We now present the incorporation of MMTS in the Transis and Horus [1] GCSs. MMTS could be similarly incorporated into other GCSs.

The main data structure of the Transis GCS is a directed acyclic graph, *DAG*, based on Trans [15] and on Psync [17]. The nodes in the DAG represent messages, and the arcs represent acknowledgments. The DAG is used to provide various reliable multicast services. We extend the DAG data structure of Transis to incorporate Multimedia Multicast Transport Service. The *begin-bunch* and *end-bunch* messages are multicast using the DAG. The regular bunch'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 **Fig. 5**(a). The Transis DAG delivers GCS messages in one dimension, and the bunch's messages are delivered in another dimension, after the corresponding *begin-bunch* message is delivered and before the *end-bunch* message is delivered.

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. MMTS may be easily incorporated in Horus. The service opens several connections to Horus as depicted in Figure **5**(b):

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



(a) The MMTS 3-D DAG in Transis (b) MMTS in Horus

Fig. 5 MMTS Incorporation in Group Communication Systems

- One unreliable unordered connection (bypassing all reliable layers), for unreliable bunches.
- A separate reliable connection for each reliable bunch.

The MMTS service needs only take care of ordering bunch messages w.r.t. the corresponding *begin-bunch* and *end-bunch* messages.

6.0 Applications

In this paper, we presented the notion of Multimedia Multicast Transport Service (MMTS) that supports multiple quality of service options (QoS) in group communication systems (GCSs). Below we describe a few applications that may exploit MMTS within GCSs.

Video multicast application are concerned with more than just multicasting a stream of video. Such application also need to exchange messages for connection establishment, flow control and negotiation and re-negotiation of QoS (agreement on QoS parameters). Furthermore, it is desirable to introduce replication in order to make such applications more available and fault tolerant. Replication arises the need for load balancing, and in order to achieve fault tolerance, the servers must consistently share information. In our approach, the video is multicast in QoS bunches, and the other tasks (*e.g.* QoS negotiation, load balancing and consistent information sharing) exploit the GCS services.

We are currently implementing a *replicated video on demand server* [4], that exploits MMTS QoS bunches to transmit video. If one server crashes or detaches from a client, the other servers get an indication, and can smoothly take over. The group abstraction is used to guarantee a transparent transition: the client continues to send its request to the same logical connection (represented by a GCS group), and is unaware of the change in server behind this connection. The group abstraction together with the GCS totally ordered multicast service are used for load balancing: each client request is multicast to the group of servers, and the servers deterministically decide which of them will serve the client.

Multimedia and desktop conferencing systems are described in the survey of CSCW systems [18]. 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. The distributed agent can exploit GCS with MMTS to provide the services listed in [18], *e.g.* floor control, dynamic reconfiguration, consistent workspace replication and management, and logging the session. In the full paper [8] we discuss these services in more detail.

7.0 Concluding Remarks

The performance of an MMTS application greatly depends on the size of the bunches, 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 bunch size: if bunches are small the overhead of handling *begin-bunch* and *end-bunch* messages is high. On the other hand, large bunches are less fault tolerant: failures may cause loss of synchronization, and applications re-synchronize at the end of bunches. The longer the bunch, the longer it takes to re-synchronize. It is our hope that further work on this topic will identify bunch sizes that induce low overhead and yet provide reasonable quality of service for specific applications.

8.0 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.

9.0 References

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

[2] 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 3, pages 84–91, June 1996.

[3] 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.

[4] T. Anker, D. Breitgand, G. Chockler, D. Dolev, and I. Keidar. Reliable Video Multicast Services. In preparation.

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

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

[7] S. Y. Cheung, M. H. Ammar, and X. Li. On the Use of Destination Set Grouping to Improve Fairness in Multicast Video Distribution. In *15th International Conference on Computer Communication (Infocom'96)*, March 1996.

[8] G. Chockler, N. Huleihel, I. Keidar, and D. Dolev. Supporting Multiple Quality of Service Options with High Performance Groupware. TR 96-3, Institute of Computer Science, The Hebrew University of Jerusalem, March 1996.

[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)*, pages 68–76, May 1996.

[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] L. Mathy. Features of the ACCOPI Multimedia Transport Service. In *European Workshop on Interactive Distributed Multimedia Systems and Services (IDMS), (LNCS, 1045)*, pages 175–194, 1996.

[14] L. Mathy and O. Bonaventure. QoS Negotiation for Multicast Communications. In *Multimedia Transport and Teleservices, International COST 237 Workshop, (LNCS, 882)*, pages 199–218, November 1994.

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

[16] 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.

[17] 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.

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

[19] 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.

[20] 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.

[21] B. C. Smith. RIVL: A Resolution Independant Video Language.

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