[Sun90] V.S. Sunderam. An Inclusive Session Level Protocol for Distributed Applications. In *Proccedings of ACM SIGCOMM '90*, pages 307–314, September 1990.

[Swi88] Daniel C. Swinehart. System Support Requirements for Multi-media Workstations. In *Proccedings of the SpeechTech '88 Conference*, pages 82–83, New York, April 1988. Media Dimensions, Inc.

[Tan88] A. Tanenbaum. *Computer Networks*. Prentice Hall, Inc., second edition edition, 1988.

[Ten89] D. L. Tennenhouse. Layered multiplexing considered harmful. In *Proceedings of Protocols for High Speed Networks; IFIP WG6.1/6.4 Workshop*, May 1989.

[Top89] Claudio Topolcic. A report on the connection oriented internet protocol meeting. ST and Co-IP Working Group Mailing List, August 1989.

[Tur86] J. Turner. Design of an integrated service packet network. *IEEE Journal on Selected Areas in Communication*, pages 1373–1380, November 1986.

[Yav89] R.S. Yavatkar. *An Architecture for High-Speed Packet Switched Networks*. PhD thesis, Purdue University, August 1989. Also available as TR-898.

[Yav90] Raj Yavatkar. MCP: A Multi-Flow Conversation Protocol for Multi-media Distributed Applications. Technical Report 181-91, Department of Computer Science, University of Kentucky, 1990. submitted for publication.

[Zha89] Lixia Zhang. *A New Architecture for Packet Switching Network Protocols*. PhD thesis, Massachusetts Institute of Technology, July 1989.

[Lam78] L. Lamport. Time, clocks, and the ordering of events ina distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[Lam84] Leslie Lamport. Using Time Instead of Timeouts. *ACM Transactions on Programming Languages and Systems*, 6(2):254–280, April 1984.

[Lan86] Keith A. Lantz. An Experiment in Integrated Multimedia Conferencing. In *Proceedings of Conference on Computer-Supported Cooperative Work*, pages 267–275, December 1986.

[Lan90] Keith A. Lantz. What is in it for us(ers)?: The Portable Office and Desktop Teleconferencing. In Craig Partridge, editor, *Report of the Internet Research Steering Group Workshop on Very-High Speed Networks*, January 1990.

[LBH+90] W-H.F. Leung, T.J. Baumgartner, Y.H. Hwang, M.J. Morgan, and S-C Tu. A software architecture for workstations supporting multimedia conferencing in packet switching networks. *IEEE Journal on Selected Areas in Communications*, 8(3), April 1990.

[Lei88] Barry Leiner. Critical Issues in High Bandwidth Networking. DARPA Internet Request For Comments 1077, November 1988. Report of a working group convened by DARPA.

[LG90] T. Little and A. Ghafoor. Synchronization and Storage Models for Multimedia Objects. *IEEE Journal on Selected Areas in Communications*, 8(3):413–427, April 1990.

[Li90] Xiangxin Li. M.S. Project Report, December 1990. Department of Computer Science, University of Kentucky.

[LT90] Charles Lynn and Claudio Topolcic. Experimental internet stream protocol, version ii (st-ii). DARPA Internet Request For Comments 1190, October 1990. IETF COIP Working Group.

[Lud89] L.F. Ludwig. Multi-Media in ISDN and BISDN: A Paradigm Shift Driven by User Technology and Applications. BellCore Digest, 1989.

[Mil89] Dave L. Mills. Measured performance of the network time protocol in the internet system. Network Working Group Request for Comments: 1128, October 1989.

[Mil90] D. Mills. Network time protocol (version 2) specificationa nd implementation. Network Working Group Requests For Comments RFC 1119, July 1990.

[MP89] T. Mazraani and Guru Parulkar. Specification of a Multipoint Congram-oriented High Performance Internet Protocol. Technical Report WUCS–89–20, Department of Computer Science, Washington University, St. Louis, Missouri, 1989.

[Nic90] C. Nicolaou. An Architecture for Real-Time Multimedia Communication Systems. *IEEE Journal on Selected Areas in Communications*, 8(3):391–400, April 1990.

[PBS89] L. Peterson, N. Buchholz, and R.D. Schlichting. Preserving and using context information in interprocess communication. *ACM Transactions on Computer Systems*, 7(3):217–246, August 1989.

[Pos80] J. Postel. User Datagram Protocol, August 1980. RFC 768.

[SFB+87] Mark Stefik, Gregg Foster, Daniel Bobrow, Kenneth Kahn, Stuan Lanning, and Lucy Suchmann. Beyond the Chalkboard: Computer Support for Collaboration and Problem Solving in Meetings. *Communications of the ACM*, 30(1):32–47, January 1987.

[SG85] Sunil Sarin and Irene Greif. Computer-based Real-Time Conferencing Systems. *IEEE Computer*, 18(10):33–49, October 1985.

[SGS84] F. Schneider, D. Gries, and R. Schlichting. Fault-tolerant broadcasts. *Sci. Compt. Program.*, 4(1):1–15, March 1984.

[Str87] C. Strathmeyer. Voice/Data Integration: An Applications Perspective. *IEEE Communications Magazine*, 25(12):30–35, December 1987.

environment. For instance, our research in exploring different levels of concurrency control spawns new opportunities for research in identifying new modes of coordination and synchronization supported at transport and higher protocol layers.

References

[AEHL88] S.R. Ahuja, J.R. Ensor, D.N. Horn, and S.E. Lucco. The Rapport Multimedia Conferencing System. In *Proceedings of the 2nd IEEE Conference on Computer Workstations*, pages 52–58, March 1988.

[BJ87] Ken Birman and Thomas Joseph. Reliable communication in the presence of failures. *ACM Transactions on Computer Systems*, 5(1):47–76, Feb 1987.

[CASD84] F. Christian, H. Aghili, R. Strong, and D. Dolev. Atomic Broadcast: From simple message diffusion to Byzantine Agreement. Technical Report RJ 4540, IBM, October 1984.

[CF89] T. Crowley and H. Forsdick. MMConf: The Diamond Multimedia Conferencing System. In *IFIP WG2.7 Working Conference on Engineering for Human Computer Interaction*, August 1989.

[Che86] David R. Cheriton. VMTP: A transport protocol for the next generation of communication systems. In *SIGCOMM '86 Symposium*, pages 406–415. ACM, August 1986.

[Che87] G. Chesson. Protocol Engine Design. In *USENIX Conference Proceedings*, Phoenix, Arizona, June 1987.

[Che88] G. Chesson. XTP/PE Overview. In *Proccedings of 13th Conference on Local Computer Networks*, pages 292–296, Minneapolis, Minnesota, October 1988. IEEE Computer Society.

[Che89] Greg Chesson. XTP Protocol Definition 3.4, 1989. Protocol Engines, Incorporated, 1900 State Street, Suite D, Santa Barbara, CA.

[Cla85] D. Clark. The Structuring of Systems Using Upcalls. In *Proceedings of the 10th Symposium on Operating Systems Principles*, pages 171–180. ACM, October 1985.

[CM84] J. Chang and N. Maxemchuck. Reliable Broadcast Protocols. *ACM Transactions on Computer Systems*, 2(3):251–273, August 1984.

[Com88] Douglas Comer. *Internetworking With TCP/IP: Principles, Protocols, and Architecture*. Prentice Hall, Inc., 1988.

[CY89] D.E. Comer and R.S. Yavatkar. Flows: Performance Guarantees in Best Effort Delivery Systems. In *Proceedings of IEEE INFOCOM '89*, pages 100–109. IEEE Computer Society, April 1989.

[Dew90] Prasun Dewan. A Guide to Suite: Version 1.0. Technical Report SERC-TR-60-P, Software Engineering Research Center, Purdue University, February 1990.

[DKK90] F.R. Dix, M. Kelly, and R. W. Klessig. Access to a public switched multi-megabit data service offering. *Computer Communication Review*, July 1990.

[EGR89] C. Ellis, S. Gibbs, and G.L. Rein. Design and Use of a Group Editor. In *IFIP WG2.7 Working Conference on Enginerring for Human Computer Interaction*. North Holland, August 1989.

[Fer90] D. Ferrari. Client Requirements for Real-time Communication Services. DARPA Internet Request For Comments 1193, November 1990.

[FKLC88] R. Fish, R. Kraut, M. Leland, and M. Cohen. Quilt: a Collaborative Tool for Cooperative Writing. In *Proceedings of ACM SIGOIS Conference*, pages 30–37, 1988.

[FV90] D. Ferrari and D. Verma. A scheme for real-time channel establishment in wide-area networks. *IEEE Journal on Selected Areas in Communications*, 8(3), April 1990.

when any of the four media (text, image, voice, and video) are used. Third, the conversation abstraction allows an application to dynamically combine any number of flows into a logical entity. Thus, an application may add or remove flows from a conversation as and when necessary to allow more concurrent communication when necessary. The latter two aspects of synchronization are not addressed by Nicolaou.

6 Summary

In this paper, we have presented a new transport protocol called MCP that provides abstractions necessary to build multimedia distributed applications. To achieve coordination in a multipoint flow, MCP provides a token-based mechanism that allows an application to exercise flexible governance over the amount of concurrency control desired. To achieve causal synchronization across traffic over multiple flows, MCP includes an abstraction called a *multi-flow conversation*. Because traffic delivery over constituent connections (such as voice or video) may have delay constraints, causal ordering is guaranteed only over an interval Δ determined by delay constraints of constituent connections.

We have built a prototype implementation of MCP using XTP and are now adding it to a Unix kernel. Initial experiments in building distributed multi-flow applications have provided encouraging results. We plan to use MCP to build a distributed interactive environment for collaboration using multiple media.

Our work has significance in the following areas. First, our research addresses an important and as yet unexplored aspect of designing suitable communication abstractions to build multimedia applications. We plan to extend results of our research to design programming language constructs that would simplify writing distributed multimedia applications. Second, our work also addresses an often neglected aspect, that of transport-level interfaces to the user applications. A research working group convened by DARPA [Lei88] has argued for richer transport-layer interfaces in next-generation networks. Development of interactive, multimedia applications using both synchronous and asynchronous transport-level interfaces will help uncover alternate, transport-level interfaces. Third, there is a considerable interest in building applications to foster collaboration and cooperation among scientists and researchers. Our work makes a significant contribution by investigating new communication methods that support collaborative activities in an interactive, distributed usually provides reliable multicast delivery.

The ISIS system at Cornell provides an asynchronous communication primitive based on "causal broadcasts". Given a group of processes concurrently broadcasting messages to other members of the group, causal broadcast guarantees that all the processes will receive messages in the same predetermined, partial order. The emphasis in ISIS is on providing fault-tolerant and globally consistent communication in the presence of site and process failures and the aim is to support consistent updates to replicated, shared objects.

The Psynch [PBS89] protocol developed at the University of Arizona is closely related to the ISIS approach. It is based on a "conversation" abstraction that defines a partial order on the delivery of messages similar to the causal broadcast in ISIS. Our approach shares some ideas with the Psynch approach where context information is represented by a global context graph maintained by each site. Context information is included in each message and a the protocol maintains a copy of global graph on each host. As explained earlier, a context graph is created independently by each site in our system and edges in such a graph are delay-dependent.

Little and Ghafoor [LG90] have suggested a formal method for specifying the temporal relationships among multimedia objects. They use timed petri nets to specify the temporal characteristics among multimedia elements stored in a data base and present an algorithm to retrieve such stored elements in a manner that preserves the original temporal relationships. However, they do not consider the problem of maintaining temporal synchronization when such elements are transmitted over independent flows.

Nicolaou [Nic90] presents a scheme for implementing synchronization among related streams originating at the same source by inserting synchronization points in each individual stream. His work is closely related to our proposal as far as achieving temporal synchronization is concerned. We allow an application to specify such points in terms of message boundaries in each individual flow. However, there are important differences between our work and Nicolaou's work. First, Nicolaou does not consider the problem of concurrency control in a collaborative system where a group of participants cooperate on a shared, multimedia document. Instead, his work is only concerned with transporting real-time voice and video in a digital network without disturbing the temporal relationship specified by a source. Second, we provide mechanisms to achieve both temporal synchronization among related streams and concurrency control among multiple participants

5.1 Communication Architecture

In [LBH$^+$90], Leung et al. propose a software architecture for workstations to support multimedia conferencing. In this architecture, a single "multimedia virtual circuit (MVC)" is used to combine traffic from various media and multiplex it onto a single, network-layer virtual circuit with variable bandwidth. The rationale for using this approach is to exploit the inherent, sequenced delivery over a virtual circuit. Use of a single virtual circuit helps ensure that all packets arrive at the receiver in the order in which they were sent.

However, this approach suffers from the following drawbacks. First, the network layer abstractions supported by ATM or other high-speed networks use a "Type of service" (TOS) parameter and a bandwidth specification (amount of bandwidth that must be reserved) to guarantee performance. Because video and voice have different TOS characteristics and bandwidth requirements, multiplexing them over a single network layer flow may lead to inefficiencies and will not allow the MVC architecture to exploit medium-specific characteristics. For example, acceptable error rates for voice (up to 1% provided an error burst is shorter than 4 milliseconds) and video (acceptable error rate depends on the compression and coding techniques used) are entirely different. Second, the complexity of managing traffic at both the sender and receiver is higher if different coding techniques are used for voice and video components of a MVC.

Our approach significantly differs from theirs because we use separate network-level flows to carry traffic belonging to different media and enforce synchronization at the transport layer.

5.2 Specification of Synchronization and Temporal Relationships

In the area of distributed systems, some researchers have designed group broadcast primitives to maintain globally consistent order of delivery when messages are broadcast among a group of processes. Examples of such primitives include an atomic broadcast [CM84, CASD84], a reliable broadcast [SGS84], and a causal broadcast [BJ87, PBS89].

In the case of atomic broadcasts, consistent delivery is achieved either by making a group member responsible to establish an order among broadcasts [CM84] or by delaying delivery of a message for a period bounded by the granularity of clock synchronization and the intersite packet delivery time [CASD84]. Overhead and complexity in each case makes these approaches unsuitable for real-time communication where atomicity is not the primary objective and the network layer specifies the upcall procedure to be called when one of pre-defined status changes occurs on a conversation. The arguments to the `procname` include conversation identifier, an integer code specifying the kind of change, relevant flow identifiers (if any), and the user data received. A similar primitive is also provided for a flow.

Such an asynchronous interface is useful for a multimedia application that must simultaneously handle incoming traffic and changes in status of multiple connections. For example, a shared image-display system may specify a routine to be called for automatic updating of the screen when data is received on a connection that only handles remote mouse and keyboard events.

4.3 Current Status

Currently, we have implemented MCP interface as a collection of library routines on top of the XTP kernel implementation on Sparcstations. The prototype has served two purposes. First, it forced us to specify the protocol completely and helped us uncover some design bugs. Second, we have used the prototype to build a distributed multimedia window-based application involving voice and text (keystroke and mouse events) flows. The traffic over the flows is simulated using bursty sources of traffic that model packet-voice traffic and the interactive sessions. To verify the performance advantages of MCP, we have also built the same applications using TCP (Transmission Control Protocol) as the transport protocol where application layer exercises token-based control and causal delivery. The latter version not only performs slowly, but also demonstrates the difficulties in achieving coordination and temporal synchronization at the application layer. We will include the results of our performance evaluation in the final version of this paper.

We are now moving the MCP implementation into the Unix kernel and plan to integrate it with the Unix socket interface for use with both XTP and Co-IP. Work is also in progress in extending Suite [Dew90] to build PolySchmues.

5 Related Work

Our work is related to current research in two main areas: communication architecture and specification of synchronization and temporal relationships.

4.2.4 Token Management

MCP also provides service primitives for replicating, distributing, transferring and deleting a token. An application can create multiple copies of a token by using the primitive `replicate_token(<token>, <# copies>)` and distribute a replicated token using `distribute_token(<token>, <# participants>, <list of participants>)`. The MCP provider permits token distribution only if appropriate number of copies of the specified token exist locally.

Sometimes, it may be necessary to divide a discussion group (a multipoint flow) into smaller, independent discussion groups. In such a case, controlling token for the flow must be explicitly transferred to each group so that each group has its own concurrency control. The service primitive `transfer_token(<token>, <# participants>, <list of participants>)` must be used for this purpose.

Finally, a copy of the token may be destroyed at a site using the primitive `delete_token(<token>)`. However, the original controlling token for a flow always remains in existence and is only destroyed when the flow is terminated.

4.2.5 Other Features

An interesting feature of MCP is the use of an out-of-band signaling channel for exchanging control information to manage a flow or a conversation. Because the transmission and delivery of data over a multipoint flow is controlled by transfer of tokens and relevant control information, processing of normal data is complicated if one uses "in-band signaling" as in traditional transport protocols. Under in-band signaling, control information related to management of a flow is inserted in a data stream. Also, additional control information must be inserted in user data to enforce causal relationship among flows in a conversation. To facilitate real-time protocol processing and to simplify processing of user data, we separate the control information flow from normal data transfer using a separate signaling channel.

Another interesting feature is an upcall-based [Cla85] user interface to allow asynchronous, event-triggered actions. Under such an interface, an MCP user can specify an action to be taken on receipt of data over a flow or in case of an event such as a flow joining or leaving a connection. The service primitive `conv_status(<status change type>, <procname>)`

The `init_flow` primitive only creates a flow at all participating sites. Each participant must explicitly invoke the primitive `accept_flow(<flow name>)` before receiving and sending any traffic over the flow. `accept_flow` returns a flow identifier and (optionally) a token if the creator has requested distribution of the token to that participant. Flow termination is achieved by each participant using a `terminate_flow` primitive.

4.2.3 Conversation Management

The service primitive `init_conv(<# flows>, <list of flow identifiers>)` creates a conversation. The list of constituent flows may be empty as flows can be added to a conversation later. `Init_conv` is symmetric and requires all the participants to issue the primitive `init_conv` before a conversation can be used. The primitive `init_conv` returns a conversation identifier to the requesting participant and is used in subsequent conversation-specific primitives. The tokens from constituent flows are implicitly transferred to the conversation at each holding site and are then managed as part of a conversation.

The primitives `join_conv(<conv identifier>, <flow id>)` and `leave_conv(<conv identifier>, <flow id>)` are used for adding and removing a flow to and from a conversation.

Any participant may issue the primitive `join_conv`; the local conversation provider adds the requested flow to the conversation and forwards that request to its peers at other participating sites. The local conversation provider will also start exchanging control information with its peers to enforce temporal synchronization for messages sent over the newly added flow and other constituent flows. However, at other peer sites, the local participant must explicitly issue the primitive `join_conv` before temporal synchronization is enforced for outgoing messages over the requested flow. This restriction is necessary to avoid confusion resulting from delayed delivery of earlier flow traffic at some sites.

Likewise, the primitive `leave_conv` simply results in removal of the requested flow from the conversation at that site. The local conversation provider informs its peers about the change. Each participant must explicitly issue the primitive `leave_conv` at each site before outgoing traffic over the requested flow is not considered subject to temporal synchronization.

(eXpress Transfer Protocol) is a lightweight, transfer[2] layer protocol being developed by a group of researchers and developers at Protocol Engines Incorporated (PEI) [Che88, Che87, Che89]. XTP offers many services, including real-time datagrams, reliable multicast data transfer, and efficient bulk data transfer. It allows an application to specify the kind of error control needed and uses rate control to maintain delivery rates suitable for real-time applications.

Our MCP design is based on XTP[3]. MCP extends the services provided by XTP to provide the conversation abstraction, multipoint flows, and token management. In the following, we describe the services provided by MCP. Our discussion omits the details of protocol headers, addressing structure, and values of various parameters to keep the exposition clear and relevant.

4.2.1 MCP Services

MCP provides three kinds of services, namely, flow management, conversation management, and token management. A MCP provider resides at each site and interacts with its peers residing at other sites to coordinate management of flows and conversations. Applications interact with a MCP provider through a set of service primitives.

4.2.2 Flow Management

Under flow management, MCP provides service primitives for flow creation, flow termination, and primitives for sending and receiving messages. An application establishes a flow by using the service call

```
init_flow(<flow name>, <flow parameters>, <# participants>, <list of participants>)
```

The `init_flow` returns a flow identifier (a `flow_id`) and a token for coordinating data transfer. The user-specified (that is agreed upon in advance) "flow name" (in addition to the flow identifier) serves to identify a flow at each participating site. One or more participants are specified using a transport-level multicast or unicast address, and flow parameters include performance (delay constraints and bandwidth requirements) and error control specifications.

[2]The transfer layer is formed by combining the functionalities of both network and transport layers of the ISO OSI model into a single layer.

[3]The author is a research affiliate of PEI and has access to the XTP kernel sources.

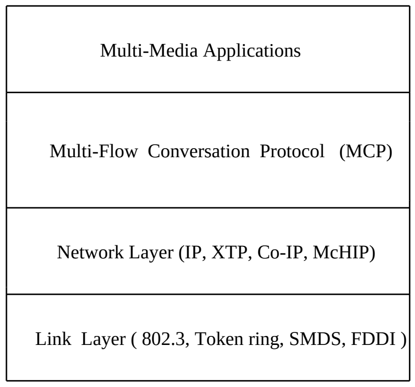
Multi-Media Applications
Multi-Flow Conversation Protocol (MCP)
Network Layer (IP, XTP, Co-IP, McHIP)
Link Layer (802.3, Token ring, SMDS, FDDI)

Figure 2: Protocol layering in the proposed communication architecture.

semantics, and UDP supports an unreliable datagram facility. Subtle interactions of buffer management and multiplexing at the session layer with those functions at the transport level and duplication of some functions (flow control, sequencing and preserving message boundaries, and buffer management) at the session layer led to poor performance.

Some real-time applications such as video are sensitive to the jitter (greater delay in delivery of traffic than the maximum allowed for a connection) caused by processing delays arising due to interactions of buffer management, layering, and scheduling operations. Tennenhouse [Ten89] makes a strong case for avoiding unnecessary multiplexing in additional layers of the protocol stack to reduce such delays.

Based on these observations and our initial experience, we decided to design a new transport level protocol.

4.2 Protocol Details

Figure 2 shows the proposed communication architecture based on layering in the DARPA Internet protocol suite. At the network layer, we expect to have both connectionless service (IP) as well as a connection-oriented service such as that provided by Co-IP (Connection-oriented internet protocol) [Top89, LT90], McHIP (Multipoint Connection-oriented High Performance Internet Protocol) [MP89], or XTP [Che88]. Both Co-IP and McHIP support establishment of connections with specific performance requirements such as bounds on delay and required data rates. Both the protocols are still under development and their implementations are not yet available to us. XTP

Proposition 1. *If individual flows meet flow delay constraints, MCP ensures delivery of messages in* Δ*-causal* ($\xrightarrow{\Delta}$) *order.*

The parameter Δ acts essentially as a tuning knob; when Δ is zero a conversation only guarantees causal synchronization among messages originating from the same sender and when Δ goes to infinity you get strict causality similar to causal broadcast in ISIS. The latter is useful when dealing with non real-time ("batch") multimedia applications such as mail or a slide presentation playback where a remote server is sending data from a multimedia storage.

Note that the idea of using and preserving context information is not completely new as it has been used before in distributed systems [PBS89, BJ87]. However, we have extended it in two directions. First, we use the context information across traffic over multiple flows in a conversation rather than treating it separately within each individual flow. Second, the notion of causality provided is weaker and bound by real-time traffic delivery constraints as demanded by multimedia distributed applications.

4 Multi-Flow Conversation Protocol

4.1 Why do we need a new transport protocol?

An important research question is what layer of protocol stack can provide the necessary semantics of multi-flow conversations and token-based concurrency control. Both the conversation abstraction and token-based concurrency control are end-to-end, abstract functions not found in existing transport protocols such as TCP [Com88], UDP [Pos80], ISO/TP4 [Tan88], and VMTP [Che86]. These features also involve one or more transport level connections spanning multiple sites. Thus, it seems appropriate to incorporate them in a session layer protocol.

We first explored this possibility by designing a session layer protocol. However, we uncovered some significant performance and implementation problems [Li90].

Implementation of conversations as well as the token-based mechanism over traditional protocols requires one to implement additional flow control, a scheduling mechanism for traffic over constituent connections, session-level buffering, and sequencing at each participating site. Also, the session layer protocol must hide the incompatible semantics of the underlying transport protocols. For example, TCP provides a reliable byte stream, VMTP supports a request-response message

The purpose of the context is to provide enough information so that receiving sites will deliver message M_1 in a causal order with respect to messages received at S_1 before sending M_1.

3. The context information included with each message is a pair `<sender_id, mesg_id>` for each constituent flow of the conversation. That is, the context includes the sequence number of the last message received at S_1 from each participant before sending M_1. Thus, the amount of context is bounded by the number of conversation participants.

   However, not all the context information may be current and useful if last message from some participant was not received in recent past. In particular, causality interval implies that a message M received earlier at S_1 provides no context for M_1 if M was received at time less than $T - \Delta$ if T is the time at which M_1 is sent by S_1. Thus, the amount of context preserved with a message is also bounded by the causality interval Δ.

4. When a receiver R_1 at another site receives message M_1, it checks to see whether or not it has already received all the messages specified in the context of M_1. If not, it buffers M_1 waiting for the missing messages to be delivered and notes the timestamp T_{m1} for M_1. It may have to buffer additional messages in M_1's context if they also arrive out of order. However, if the missing messages are not received within the interval Δ after T_{m1}, the conversation provider delivers M_1 to its application and deallocates buffers.

5. As R_1 receives messages in a conversation from different sites, the context information in each of those messages defines a Δ-causal order among some or all of those messages in terms of in what order should they be delivered to the application. In fact, R_1 constructs a directed graph whose nodes are messages and an edge from M_1 to M_2 specifies that M_1 must "precede" M_2. However, structure of the graph and edges in it are deleted as the clock ticks and some context becomes irrelevant.

   Moreover, the context graphs constructed at two different sites R_1 and R_2 involving the same set of messages and conversation can be different depending on when and in what order each of those messages actually gets delivered.

Given Definition 3 and the algorithm stated above for message delivery, it is easy to see the following result.

$$\Delta_1 = max\{\delta_i, i = 1, \ldots n\} \qquad \Delta_2 = min\{\lambda_i, i = 1, \ldots n\}$$

Considering flows over different media including voice, video, image, and text, we find that Δ_2 is usually larger than Δ_1 by a factor of at least two.

Based on these two constraints, we define the causality interval Δ for each conversation. Δ is computed as:

$$\Delta_1 < \Delta < \Delta_2$$

and defines a window of causality for each message sent in a conversation. For example, if a message M_4 is sent by a participant after receiving messages M_1 through M_3 over some constituent flows, M_4 must normally be received by applications at all other sites in the correct causal order (i.e., after M_1 through M_3 are received). However, Δ-causality specifies that a conversation provider at each receiving site maintain such causal relationship between M_4 and each of its predecessors only if messages related to M_4 are received within interval Δ.

It must be noted that our notion of real-time assumes large-grain clock synchronization among the participants. This is not an unrealistic assumption as fault-tolerant clock synchronization algorithms exist that achieve such synchronization [Lam84]. In TCP/IP Internet, Network Time Protocol (NTP) achieves global clock synchronization across the country within a few milliseconds [Mil90, Mil89]. Thus a small upper bound ϵ (typically 2 or 3 milliseconds) on difference between clocks at any two participants can be assumed.

The value of Δ is chosen based on the following pragmatic considerations. First, Δ must at least be equal to $\Delta_1 + \epsilon$. Second, to allow some flexibility in the presence of fluctuations in network conditions and delays, MCP provider may choose value of Δ to be higher than $\Delta_1 + \epsilon$ without compromising individual flow semantics as long as Δ remains much below the upper bound Δ_2.

The interval Δ is used as follows:

1. Each conversation is assigned a network-wide unique conversation identifier `conv_id`; each message in a conversation is timestamped and is assigned a conversation-wide unique `mesg_id`[1].

2. Whenever a participant at a site S_1 sends a message M_1 over one of the constituent flows of a conversation, the conversation provider at that site includes some *context* for that message.

[1]A conversation-wide unique message identifier is easily constructed by appending a local sequence number to the address of the participating site.

establish a voice connection among all the users. It will also establish a reliable data flow among the users that will transfer all mouse and keyboard events. It will then create a conversation and add the two flows to the conversation to achieve the desired semantics of "free-for-all" coordination. However, if it decides to enforce a stronger level of coordination as in the "floor-control" coordination, it need only restrict the possession of tokens to a single participant at any time.

The composition of a conversation is not fixed at the time of its creation; flows may be added to and removed from a conversation at any time. Thus, an application can exert dynamic control over the degree of synchronization needed by simply excluding certain flows from the conversation whenever necessary. For example, consider a user in a collaborative, software engineering environment annotating parts of a text when another user is simply commenting on or discussing a design document. In such a scenario, not all the flows involved need to be part of the same conversation all the time.

3.3 Delta-Causality

In the following, we describe the notion of Delta-Causality in detail.

We assume that each flow has performance characteristics associated with it that are specified when a flow is established. Apart from the bandwidth and error rate parameters [CY89, DKK90], there are two delay constraints associated with each flow F_i:

Desired delay δ_i is the maximum end-to-end delay that the flow F_i can tolerate before quality of service deteriorates. Example of such a constraint is 100 milliseconds (ms) delay bound in packet voice.

Loss delay λ_i is an upper limit on end-to-end delay for flow traffic beyond which the delivered traffic delivery is useless. For example, packet voice or video have *better-never-than-late* delivery semantics and specify such a constraint. Packet voice traffic with desired delay constraint of 100 ms has a loss delay constraint in the range of 200 to 300 ms. Among voice, video, and image traffic, packet voice has probably the most stringent loss delay limit. For other flows, loss delay constraint is typically a larger multiple of desired delay constraint.

Consider a conversation C consisting of flows $F_1 \ldots F_i \ldots F_n$ with respective delay constraints δ_i's and λ_i's. We compute two conversation-wide desired delay and loss delay constraints:

2. m_2 is sent over some flow f_i by sender S_1 after receiving m_1 over some flow f_j, and both $f_i, f_j \in F$

Thus, *Definition 2* defines a partial, causal order among all the messages exchanged over a conversation. However, enforcement of such a causal order among messages sent over a conversation has performance penalties (including delaying message delivery for a long time) associated with it and such penalties are not acceptable to real-time flows involving voice or video that have *better-never-than-late* semantics. For instance, real-time voice flows must deliver traffic within 100 milliseconds before quality of service degrades and traffic delivered after about 300 milliseconds is useless for voice interaction.

In fact, delivery of certain messages might be meaningless if delayed beyond the delay constraint of the flow involved. Therefore, we have decided to discard strictly causal ordering in favor of a notion of causality called **Δ-causality** that takes into account the delay constraints associated with constituent flows.

Informally, Δ-*causality* works as follows. Given a set of flows and a set of participants in a conversation, message delivery to an application at a recipient site is causally ordered provided an upper bound Δ on end-to-end delay is not violated for any of the related messages. For instance, in the example described above, the receiver S_3 will see the proper causal order provided M_1 and M_2 were generated and sent in such a way that their arrival at S_3 is not delayed beyond an upper bound Δ. Δ is a function of individual delay constraints for flows $Flow_1$ and $Flow_2$.

To take delay constraints into account, we define a Δ-Causal order as:

Definition 3. $m_1 \overset{\Delta}{\rightarrow} m_2$ if :

1. $m_1 \rightarrow m_2$ and

2. m_1 is sent at most Δ time units before m_2

We defer details on relationship between delay constraints of individual flows and parameter Δ until next section. We find the weaker notion of delta-causality sufficient because applications that need to enforce a stronger causal relationship among messages from multiple senders can achieve that effect using the token-based control across all the flows in a conversation.

Thus, a multi-user, shared window based collaborative application will create a voice flow to or more (two-party or multipoint) flows. An application will typically first create individual flows with appropriate performance requirements [Fer90]. It will then create a conversation that includes one or more related flows to achieve the necessary temporal synchronization.

If a conversation C (see Figure 1) consists of two flows, $Flow_1$ and $Flow_2$, and if a sender S_1 sends a message (some data) over $Flow_2$ and then sends another message (some data) over $Flow_1$, all the participants should receive those messages in the same sequence.

Formally, let F denote the set of flows in a conversation C and let M denote the set of messages sent over the constituent flows in F.

Definition 1. We define $\prec$ ("precedes") to be a transitive relation on M, such that $m_1 \prec m_2$ if and only if the following conditions hold:

1. both m_1 and m_2 are sent by the same sender, and
2. both m_1 and m_2 are sent over flows f_i and f_j (both $f_i, f_j \in F$), and
3. the message m_1 is sent before m_2 is sent.

Ideally, each participant in a conversation must receive messages in the partial order defined by the relation $\prec$.

This notion of causality is limited and does not necessarily provide causally ordered delivery of messages originating from multiple senders. For example, if a sender S_1 sends a message M_1 over a constituent flow $Flow_1$ and a sender S_2 sends a message M_2 over another constituent flow $Flow_2$ **after** receiving M_1, some other participant (say, a receiver at site S_3) may not receive messages in the causal sequence (M_1 followed by M_2).

Stricter notions of causality enforced in transaction-based systems such as an atomic broadcast [CM84, CASD84, SGS84] or a causal broadcast [Lam78, BJ87, PBS89] achieve a global ordering of messages necessary for this purpose.

Following [Lam78], we define a *causal dependency* relation "$\rightarrow$" among messages in a conversation as: :

Definition 2. $m_1 \rightarrow m_2$ if and only if one the following two conditions hold:

1. $m_1 \prec m_2$, or

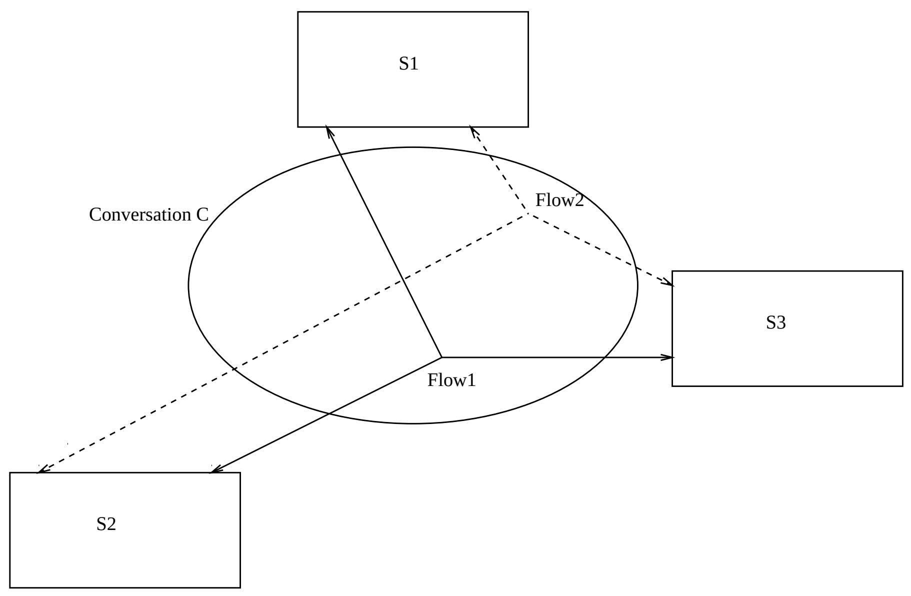

Figure 1: Conversation C consists of two flows $Flow_1$ and $Flow_2$ that span three participants S_1, S_2, and S_3.

environment interact in two phases. In the first phase, a speaker addresses a group of listeners with no interruptions. The application may switch to the second phase any time. In the second phase, a group of questioners may address questions to the speaker.

The token-based method can accommodate both types of interaction. Only the speaker may hold the token during the first phase, whereas the token may be replicated and distributed to the questioners during the second phase. Replicated tokens will be destroyed at the end of the second phase.

Discussion Groups Some environments such as real-time conferencing systems [CF89, SG85] envisage a session breaking up into smaller discussion groups and thus holding multiple, concurrent conversations. In such a system, initially only a single token would be created and passed on from one speaker to another to achieve strict "floor-based" control. However, the token can be replicated and transferred to each discussion group, and each group may then use the token independently as it sees fit.

3.2 Multi-flow Conversations

To allow temporal synchronization among traffic over multiple flows, we introduce a communication abstraction called a *multi-flow conversation*. A conversation is a logical entity and consists of one

3 Communication Abstractions

To facilitate coordination among participants of a multipoint flow and to achieve temporal synchronization among traffic over multiple flows, we propose two methods of coordination: token-based control and an abstraction called *multi-flow conversation*. Together, these two methods yield a flexible and adaptable coordination mechanism as discussed below.

3.1 Token-based Concurrency Control

When a multipoint flow is created, a token is assigned to that flow that acts as an authorization for data transmission. A sender must hold a token to be able to send traffic over a flow. However, token management primitives are provided so that other participants can obtain transmission privileges. Thus, applications are free to transfer, replicate, and delete tokens to govern the degree of concurrency control needed.

This type of concurrency control is entirely different from the token-based synchronization provided by some session-layer protocols [Tan88, Sun90]. The latter method of synchronization allows session participants to insert resynchronization points (or checkpoints) in the data stream to allow rollback and to reduce the amount of retransmitted data in case of a transmission error.

We use the token mechanism to allow several levels of concurrency control. The following control hierarchy is derived based on schemes proposed in the literature [SFB$^+$87, AEHL88, FKLC88, EGR89, CF89, SG85, Lan86, Lan90, Swi88, Str87].

Floor Control As described earlier, real-time teleconferencing systems employ such a strict concurrency control. A single token enforces such control over a multipoint connection. The token will be passed on from one speaker to another whenever the floor is transferred. To obtain control of the floor, a participant must explicitly request transfer of the token by invoking a token management primitive.

Brainstorming This form of coordination is common in shared window systems [SFB$^+$87, AEHL88, FKLC88] where there is no concurrency control for simultaneous actions by multiple users.

For such applications, the token is replicated and distributed to all the participants.

Chalkboard Interaction Applications based on the "chalkboard" metaphor in a cooperative

2.1 Degree Of Coordination

The amount of coordination needed varies in both inter- and intra-flow cases depending on the application.

In the case of a shared window package [SFB$^+$87, AEHL88, FKLC88], a single window is displayed on the display screens of multiple users, and each user gets an identical image of the window. Shared window systems provide no concurrency control for simultaneous actions by multiple users. Instead, they allow users to constantly see the actions of other users who are responsible for manually ensuring that there are no conflicts.

At the other extreme lie teleconferencing systems where interference is avoided by strict control based on the notion of a "floor", where only the current "speaker" that has the floor is allowed to "speak" (or transmit data) at any time.

Strict coordination is also necessary among separate flows when a group of users are pointing at a shared text window and discussing parts of text over a voice channel.

Both the extremes have their limitations. On one hand, lack of any concurrency control puts additional responsibility on application designers to resolve conflicts. On the other hand, strict floor-based control does not allow applications to exploit inherent concurrency and may sometimes unnecessarily increase latency due to the waiting involved. In addition, there is a continuum between these two extremes where varying degree of coordination may be necessary or desirable for present and future applications.

For instance, users in a conferencing system may sometimes decide to enter smaller discussion groups that may hold multiple, concurrent conversations [SFB$^+$87]. A "brainstorming" session based on the "chalkboard" metaphor [SFB$^+$87] in a cooperative environment is another example where less stringent coordination is appropriate. In the case of a multimedia-based group editor, strict coordination is not necessary when a user is browsing through a part of text while another is annotating a different part of the text.

Thus, the degree of coordination needed varies from time to time within an application and across different applications.

an application to specify event-triggered actions to facilitate fast processing of periodic data and changes to the status of multipoint connections.

The remainder of this paper describes design and implementation of MCP and its associated abstractions. Section 2 discusses the need for coordination and temporal synchronization. Section 3 describes token-based control, conversation abstraction, and the notion of Δ-causality. Section 4 describes the Multi-flow Conversation Protocol and its ongoing prototype implementation. Section 5 discusses the related work in this area and Section 6 provides a summary of our work.

2 Coordination and Temporal Synchronization Problem

In the following, we describe the problem of coordination and temporal synchronization.

Hereafter, we will refer to a single or a multipoint connection as a *flow*. We assume that the network layer provides a flow abstraction [CY89] that may have performance guarantees associated with it to provide predictable performance needed by real-time voice or video channels [Fer90]. The question of how to satisfy the real-time performance requirements of a flow is *not* the focus of discussion here; that question has been addressed by many researchers [FV90, Zha89, Tur86, Yav89]. Instead, the focus here is on research issues in using and combining such flows to build multimedia, collaborative, distributed applications. For the design of multimedia systems such as PolySchmues, one must address two aspects of coordination, namely, what sort of coordination is necessary and how much control must be exercised at the communication substrate. In our view, two kinds of coordination are necessary:

Intra-Flow Coordination Given a flow spanning a group of users, one must consider the problem of concurrency control when more than one user may be transmitting data at the same time. A common example of such coordination is avoiding interruptions and crosstalk in a multipoint voice channel. Such coordination is also necessary when communication involves a shared text document or an image.

Inter-Flow Coordination A multimedia application may need to synchronize traffic over multiple flows, each carrying traffic from a different medium (text, voice, video, or image). A collaborative, software development group effort described earlier is an example of such an interaction.

synchronization among related data streams. We are currently investigating such abstractions in our research and are using them in building **PolySchmues**, a distributed interactive environment for collaboration using multiple media.

We have designed a new transport protocol called *Multi-Flow Conversation Protocol* (MCP) [Yav90] that provides two communication abstractions:

First, MCP includes a token-based mechanism for concurrency control among participants of a multipoint connection. A token serves as an authorization for transmission in a multipoint connection. However, applications are free to exchange, replicate, and delete tokens to exercise flexible governance over the amount of concurrency control desired. The degree of concurrency control may vary from a strict "floor-based" control (useful in conventional teleconferencing) at one extreme to no control (as in existing shared window systems [SFB+87, AEHL88, FKLC88]) at the other extreme.

Second, MCP provides a communication abstraction called a *multi-flow conversation* to allow temporal synchronization among traffic over multiple, independent streams. A conversation is a logical unit of interaction and may consist of one or more (two-party or multipoint) connections.

We enforce temporal synchronization in delivery of messages sent over participant connections. For instance, consider a conversation consisting of a multipoint voice connection V_1 and a data (text) connection T_2. If a sender S_1 sends a message over T_2 followed by a message over V_1, all the conversation participants will receive those messages in the same sequence.

In addition, a conversation provides limited causal ordering among messages sent by two or more sources. For instance, if S_2 sends a message over a constituent connection after "hearing" from S_1, all other participants must "see" messages from S_1 and S_2 in the proper causal order.

Unlike the total global ordering of events defined by Lamport [Lam78], causal ordering in a conversation is restricted based on a notion of *delay-constrained* or **Δ-causality** as explained later. Because traffic delivery over constituent connections (such as voice or video) may have delay constraints, causal ordering is guaranteed only over an interval Δ determined by delay constraints of constituent connections.

Additional features of MCP include an out-of-band signaling channel for exchanging control information and an asynchronous interface to applications. Out-of-band signaling facilitates real-time protocol processing and simplifies processing of user data. The asynchronous interface allows

FKLC88]. In such an environment, a group of designers located at different sites collaborate on a design document (or a program) using interactive tools to edit and test parts of the design under development. Interaction may involve a group editor (based on shared windows), an image display (that displays resulting design), and a voice channel that allows them to view, discuss, and edit the suggestions made by each other.

We concentrate on the issues of coordination and synchronization of traffic over related data streams. These issues arise in following forms:

- A single multipoint communication involving multiple users on multiple, remote sites requires causal synchronization so that all participants "see" all the communication events in the same or "correct" order. For instance, a voice conference channel that spans multiple participants requires such a synchronization.

  Apart from voice/video interactions, both communication and application-level software must also enforce some degree of concurrency control when a group of users may simultaneously view and update shared text or image data to maintain consistency.

- The communication software must allow an application to coordinate multiple information channels that carry traffic from multiple media such as text, voice, video (or image). The collaborative, software development group effort described earlier is an example of such an interaction. For such an application, the communication software must allow related streams to be grouped together and recognize the order and sequencing of traffic sent over them. For example, when a participant scrolls through an image browser (or a shared editor window) and says, "look at the middle of the display", the statement should be heard at the same time (or just after) the scrolling is completed. Such temporal relationships must be captured in the delivery of traffic over related streams. Another example of such a synchronization is "lip-synching" when voice and video are transmitted over separate connections in a digital network.

Existing transport and/or session layer protocols lack communication abstractions that provide the necessary semantics for coordination and temporal synchronization in multimedia applications. Our aim is to identify appropriate communication abstractions that must be designed and supported so that an application can specify and achieve necessary concurrency control and temporal

MCP: A Protocol For Coordination and Temporal Synchronization in Multimedia Collaborative Applications*

Raj Yavatkar
Department of Computer Science
University of Kentucky, Lexington, KY 40506-0027

November 11, 1991

Abstract

With the advent of high-speed networks, it is possible to build multimedia distributed applications that involve a geographically dispersed group of users. Development of such applications requires support for coordination and temporal synchronization of traffic over related streams.

For instance, one must consider the problem of coordination in a multipoint communication where more than one sender may transmit data at the same time. Also, a multimedia application may need to synchronize traffic over multiple connections, each carrying traffic from a different medium. When a participant scrolls through an image browser (or a shared editor window) and says, "look at the middle of the display", the statement should be heard at the same time (or just after) the scrolling is completed. Such temporal relationships must be captured in the delivery of traffic over related flows.

Existing transport and/or session layer protocols do not explicitly include communication abstractions that provide the necessary semantics for coordination and temporal synchronization. We present a new transport protocol called *Multi-Flow Conversation Protocol* (MCP) that provides two communication abstractions. First, MCP provides a token-based mechanism for concurrency control among participants of a multipoint connection. Second, MCP includes a novel communication abstraction called a *multi-flow conversation* to allow temporal synchronization among traffic over multiple, independent streams. A conversation may consist of one or more (two-party or multipoint) connections, and MCP enforces temporal synchronization in delivery of traffic over participant connections. Delivery of traffic in a conversation is based on a notion of causality called **Δ-causality** that takes into account the delay constraints associated with real-time traffic.

We describe MCP and its associated abstractions in detail and describe an implementation of MCP based on the eXpress Transfer Protocol (XTP).

1 Introduction

With the advent of high-speed networks [DKK90, Lud89], it is now possible to build multimedia distributed applications that involve a geographically dispersed group of users. An example of such an application is a collaborative, software engineering environment [EGR89, SFB$^+$87, AEHL88,

*This research was supported in part by the National Science Foundation and by the Center for Advanced Manufacturing and Industrial Automation of the University of Kentucky.