Report Number: WUCS-99-24

1999-01-01

Floor Control Protocol for ALX Video Conference Application

Ruibiao Qiu

With wide deployment of high-speed networks such as vBNS today, video-conference applications over WANs have become increasingly feasible. MMX has proven to be a good desktop video-conference devide for local ATM networks. Now, ALX has been designed to extend MMX's video conferencing capability to IP-over-ATM WANs such as vBNS. In this report, we discuss a floor control protocol for ALX video-conference applications. We first show how an "ideal" protocol should behave to meet our requirements. Then we compare three protocols based on distributed algorithms, and a protocol based on a centralized algorithm. Based on the comparison and performance analysis,... **Read complete abstract on page 2.**

Floor Control Protocol for ALX Video Conference Application

Ruibiao Qiu

Complete Abstract:

With wide deployment of high-speed networks such as vBNS today, video-conference applications over WANs have become increasingly feasible. MMX has proven to be a good desktop video-conference devide for local ATM networks. Now, ALX has been designed to extend MMX's video conferencing capability to IP-over-ATM WANs such as vBNS. In this report, we discuss a floor control protocol for ALX video-conference applications. We first show how an "ideal" protocol should behave to meet our requirements. Then we compare three protocols based on distributed algorithms, and a protocol based on a centralized algorithm. Based on the comparison and performance analysis, we plan to implement the protocol based on the centralized algorithm for initial test-runs. We also describe what our future work will involve.

Floor Control Protocol for ALX Video Conference Application

Ruibiao Qiu

WUCS-99-24

August 1999

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
St. Louis MO 63130

Floor Control Protocol for ALX Video Conference Application

Ruibiao Qiu

Department of Computer Science
Washington University
One Brookings Drive
St. Louis, MO 63130
(ruibiao@arl.wustl.edu)

August 31, 1999

Abstract: With wide deployment of high-speed networks such as vBNS today, video-conference applications over WANs have become increasingly feasible. MMX has proven to be a good desktop video-conference device for local ATM networks. Now, ALX has been designed to extend MMX's video conferencing capability to IP-over-ATM WANs such as vBNS. In this report, we discuss a floor control protocol for ALX video-conference applications. We first show how an "ideal" protocol should behave to meet our requirements. Then we compare three protocols based on distributed algorithms, and a protocol based on a centralized algorithm. Based on the comparison and performance analysis, we plan to implement the protocol based on the centralized algorithm for initial test-runs. We also describe what our future work will involve.

1. Introduction

1.1. Background

vBNS (very high performance Backbone Network Service) [1] is a nationwide IP-over-ATM network service set up as a joint effort by the NSF and MCI to promote the collaboration among research and education institutes and provide a high bandwidth network for research applications. Operating at OC-12 bandwidth, it is based on MCI's hyperstream ATM service. As a logical IP network, vBNS has fully meshed topology.

With the wider deployment of high-speed switches and network services such as vBNS, we are able to deploy more bandwidth-demanding applications such as video-conferencing to wider areas. Building on our local video-conference applications, we can introduce an efficient and high-quality video-conference application over vBNS with MMX and ALX.

MMX (MultiMedia eXplorer) boxes are used in our video-conference applications as desk-top video/audio interface devices. Normally, MMX is used in local ATM-only environments. For video-conferences over vBNS, longer delays and bigger delay jitter may result in bad video quality. Moreover, the AAL0 cells produced by MMX boxes need to be converted to appropriate formats before they can be routed and transmitted over IP networks such as vBNS.

To overcome the problems, the ALX (Adaptation Layer Translator) has been specially designed with video-conference applications over vBNS in mind. An ALX sits between an MMX

box and an ATM switch, taking output AAL0 cells from MMX and converting them into AAL5 packets sent to the switch. To transmit these packets over IP networks such as vBNS, ALX also needs to add configurable IP headers for IP routing. At the receiver end, ALX strips IP headers and converts AAL5 packets back to AAL0 cells for MMX to display. To overcome delay jitter and obtain real-time quality video, ALX uses buffering and audio packet sequencing.

ALX can handle one video stream and two audio streams on three different virtual circuits at the same time. All other traffic is bypassed. One audio stream is associated with the video stream with the same channel ID. Audio packets are assigned sequence numbers before they are sent out. Video packets are assigned the same sequence number as the audio stream. At the receiver side, ALX takes away IP headers and AAL5 header/trailers from the packets, queues up the packets, and sends them out to MMX according to the order of sequence numbers, with constant intervals between audio packets.

The report is organized as follows. The rest of section 1 briefly introduces the architecture of our ALX video-conference application, and describes related work. Section 2 investigates how an ideal floor control protocol should behave in such a video-conference application. Section 3 explores protocols that are based on distributed algorithms. Section 4 shows a protocol based on a centralized algorithm. Finally, section 5 outlines our conclusions and provides a discussion of future works and deployment of the video-conference application in vBNS.

1.2. Video Conference Applications

A video-conference application over vBNS should meet the following requirements:

- Real-time studio quality video and audio

Delay jitter should not affect video and audio quality. With the buffering and resequencing function of ALX, the effects of delay jitter caused by vBNS should be constrained.

- Quick and reliable speaker changes

When the speaker changes, all participants should be able to see and hear the new speaker quickly and consistently.

- No simultaneous transmission on one channel

Because ALX can handle only one video and two audio streams at a time, and all packets are queued with sequence numbers, flushing video and audio will occur if two or more participants are transmitting on the same channel. Thus, the floor control mechanism should prevent this from happening.

- Scalability

The application should work fine with both small and large conferences.

1.3. Architecture of ALX video-conference application

Figure 1 shows the architecture of the ALX video-conference.

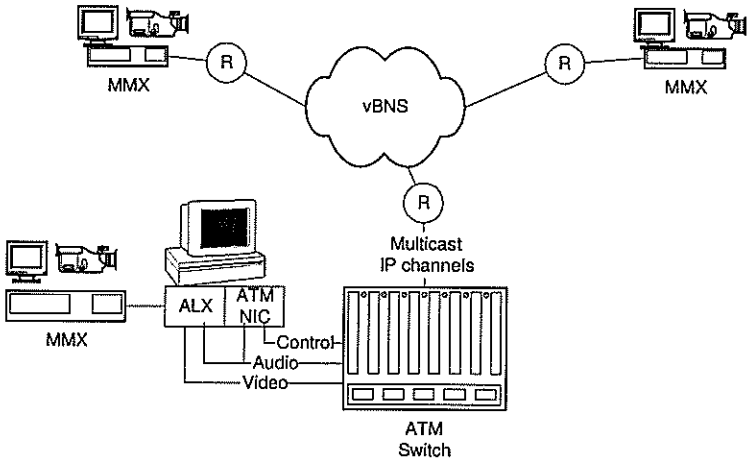


Figure 1. ALX Video-Conference Application Architecture

As shown, each participant in a conference is equipped with an MMX box with video camera and monitor. This MMX box is then daisy-chained with an ALX card in a workstation. The ALX has a direct connection to an ATM switch, which then connects through the edge router to vBNS. At the edge router, multicast IP addresses are reserved for video-conference channels. By channel, we mean one audio stream and one video stream transmitted by one participant. The number of reserved channels is determined by the protocol in use. There could be a fixed number of channels, or the channel number may grow as more participants join. At the edge router of each site, these reserved channels are mapped to virtual circuits on the ATM switch. Packets on these multicast IP channels are directed to the ATM switch with these predefined virtual circuit identifiers. The ALX is configured to processes packets on these virtual circuits. One ALX can handle one video stream and two audio streams on three virtual circuits. All other packets are bypassed or dropped. An ATM network interface card may be needed for other out-of-band control information. The ALX strips IP headers and AAL5 headers/trailers from received packets, and converts them into AAL0 cells. These converted AAL0 cells are then output to MMX for display. In return, AAL0 cells that MMX produces are taken by ALX and then transmitted with appropriate multicast IP and AAL5 encapsulation.

In this conference model, all participants receive audio and video from the current speaking participant, except the speaker himself who can receive audio and video from the participant who has spoken most recently. When a participant wants to speak, control messages are sent out over a control channel. Upon receipt of this control message, a decision is made concerning the choice of the next speaker. Once this decision is made, all participants switch to the new speaker, while the new speaker is still tuned to the speaker he replaces, who is most likely to interact with him.

1.4. Related Work

Vaudeville [1] is a video-conference application that is based on MMX developed at Washington University. Unlike our application, Vaudeville is designed to be used in a local ATM network environment. Vaudeville is developed with the "Playground" development tool for distributed systems. In Vaudeville, conference information is stored in a centralized place. Participants join a conference by connecting to and retrieving information from this conference manager. Control information is exchange via the "Playground" message mechanism. Also, Vaudeville uses a voice-activated model for speaker switching. Once a MMX detects audio input from a participant that exceeds a threshold, it begins to send that participant's video and audio. This is a

good exploitation of MMX's features. However, since Vaudeville is mainly used locally, and relies on the "Playground" mechanism to control information exchange, it is not easy to deploy it over vBNS.

Malpani and Rowe introduced a "questionboard" floor control mechanism in their MBone seminar tool call *qb* [5]. In this floor control mechanism, participants send request messages to a designated conference moderator to ask a question. These requests are shown on the moderator's user interface. He is usually the seminar lecturer, and can either select one question to answer, or grant the floor to one remote participant as a speaker. This floor control mechanism is mainly used in a seminar model conference where one participant transmits all the time, and other participants can ask questions by typing. Unfortunately, this is not suitable for our conference model, where every participant has an equal chance and need of speaking.

The Conference Control Channel Protocol (CCCP), developed by Handley, Singhal, and Cheriton [6], presents a scheme of conference control. CCCP uses a common conference control channel to facilitate conference control message exchange among applications. This is like the control channel we use. All conference management, including floor management, is done through messages exchanged in this channel by applications. CCCP gives a general scheme for all kinds of conferences. However, the application developers still need to define control messages and responses for these control messages.

The floor control mechanism in the Simple Conference Control Protocol (SCCP), developed by Bormann, Ott, and Reichert [7], uses a centralized conference moderator to grant permission to the next speaker. SCCP assumes a reliable underlying transportation mechanism, and does not address cases of message loss.

2. Ideal Floor Control Protocol

Imagine an ideal floor control protocol for ALX video-conference over vBNS. How should it behave? A discussion of such a protocol will help us to develop a real protocol.

- First of all, an ideal floor control protocol should be able to switch speakers reliably and quickly. Once a new speaker is granted the permission to transmit, all other participants should be informed of the change, and be tuned to his video and audio immediately. When a new participant joins the conference, he should be tuned to the current speaker quickly.
- An ideal protocol should scale easily over vBNS without much resource consumption. When a conference is small, there should not be many resources reserved but not used. When a conference grows bigger and bigger, resources consumption should not quickly exceed available resources.
- An ideal protocol should be able to deploy to participant sites. It should not rely on a specific local environment or installed software.
- An ideal protocol should manage conference participants automatically and dynamically. When a participant joins or departs a conference, all others should see the change.
- An ideal protocol should be able to handle all possible faulty situations. These failures include message loss, link failure, and host failure. In case of any of these failures, the protocol should detect it and recover to a normal state.

According to these principles, an ideal floor control protocol should operate as follows:

At startup, a participant can find information of current on-going conferences from a control information channel. Once a new participant joins a conference, his joining should be reflected in the conference information immediately.

When a participant wants to speak to other participants, he can take the floor and begin to speak without a long wait. When speaker change occurs in a conference, all participants should be tuned to the most current speaker quickly and correctly. Once a participant wants to speak, he can always find an available channel without conflict and delay. Once a participant notices the change of speakers, and is switched to the new speaker's channel, he should be able to see and hear the new speaker without any delay. Participants send their requests by clicking buttons on their user interfaces.

In case of possible failures, conferences should not be affected. These possible failures include link failure and host failure. If a link fails, a participant should be able to get conference information continuously unless he loses all connections to vBNS. If a participant's host fails, all other participants in a conference should be able to continue without any disturbance. If a current speaker fails, a new speaker should be picked to continue with the conference.

3. Protocols based on Distributed Algorithms

The protocols introduced in this section are all based on distributed algorithms. There is no centralized controller for conference control and information exchange. Advantages of such protocols include shorter delay and elimination of any single point of failure. A major drawback is their complicated control mechanism.

In the following sections, we use **current speaker** to denote the participant who is currently speaking to all other participants, and we use **previous speaker** to denote the participant who was the last current speaker. Also, we refer to the audio and video channel that the current speaker is transmitting on as the **current channel**, and the channel used by the previous speaker for audio and video transmission as the **previous channel**. Control messages are exchanged on another channel, referred as the **control channel**. These channels are multicast IP addresses. We can denote them as IP_0, IP_1, etc.

Normally, all participants can hear and see the current speaker on the current channel except the current speaker himself. Instead, the current speaker can see and hear the previous speaker on the previous channel.

3.1. Two-channel Algorithm (Senatorial Algorithm)

The first algorithm uses two channels: the current channel (CS channel) and previous channel (PS channel), as shown. These two channels are fixed so that the current speaker always transmits on the CS channel, and previous speaker always transmits on the PS channel.

Figure 2 is the network diagram for a protocol based on a senatorial two-channel algorithm.

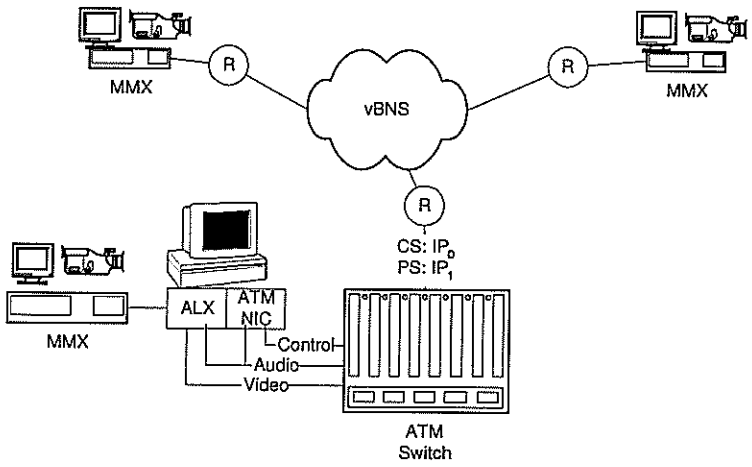


Figure 2. Senatorial Algorithm

In this algorithm, since there are only two channels, we have to make sure the channels are ready before someone can start transmitting on that channel. A participant sends a REQ (request) message to the current speaker to request the floor. If the requestor is a listener, he then waits for a REQ ACK (request acknowledgement) message from the current speaker before he can transmit on the CS channel. When the current speaker receives a request from a listener, a REL (release) message is sent to the previous speaker to request the previous speaker to stop his transmission and release the PS channel. Once the previous speaker gets this request, his transmission is stopped. A REL ACK (release acknowledgement) message is sent back to the current speaker, and the previous speaker becomes a normal listener.

When REL ACK reaches the current speaker, he knows that the PS channel is ready. He is switched to the PS channel, and a REQ ACK message is sent to the requestor, indicating the CS channel is now ready. However, he does not become the new previous speaker right away. Instead, he waits for the first ANN (announcement) message as an indication of the new current speaker before he becomes the new previous speaker. When the requestor receives REQ ACK, he becomes the new current speaker and begins transmitting on the CS channel. As a current speaker, he periodically broadcasts an ANN message to all participants to inform them of the current conference setting. When this ANN message reaches the old current speaker, he then becomes the new previous speaker. This finishes a normal round of speaker change.

In some cases, the previous speaker may want to request the floor again. In these cases, the previous speaker stops his transmission on the PS channel first, and sends his REQ to the current speaker. As the current speaker receives this REQ from the previous speaker, his channel is switched to the PS channel, and responds with REQ ACK. Upon receipt of this REQ ACK, the previous speaker becomes the new current speaker, and transmits on the CS channel. When there is a long period of time without any ANN message, it is a good indication that the current speaker is not reachable. To avoid a black screen, the previous speaker becomes the current speaker. This period of waiting time is significantly longer than the interval for ANN messages. A timeout and retransmit mechanisms are used for all control messages to avoid the impacts of message loss. Figure 3 shows the state transition diagram of the protocol based on this algorithm.

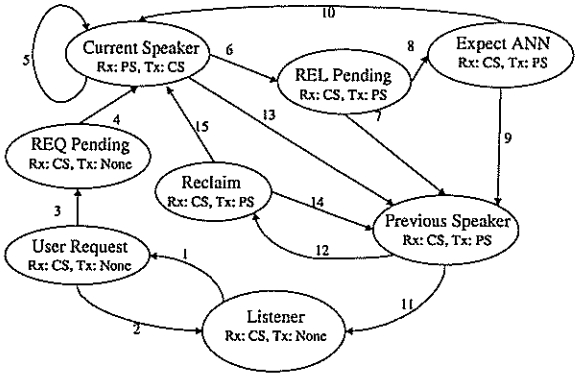
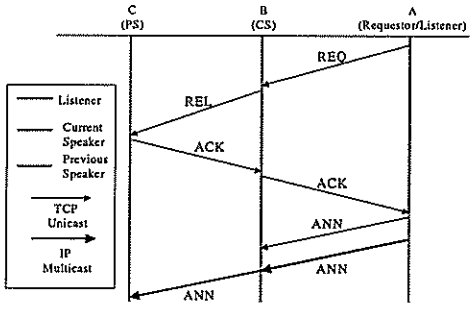


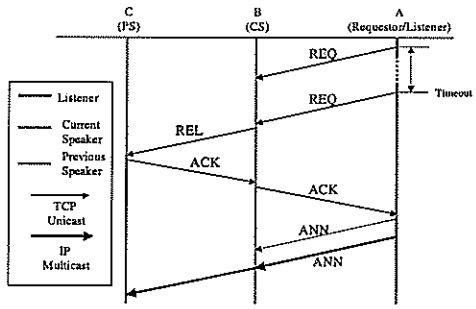
Figure 3. State Transition Diagram for Senatorial Algorithm

Number	Events	Actions
1	Button Pressed	Tries to connect to CS via TCP
2	Connection Failed	Shows error
3	Connected to CS	Sends REQ to CS
4	REQ ACK received	Becomes CS, and begins transmission
5	Clock Tick	Sends out multicast ANN to all listeners, and TCP ANN to CS
6	REQ from a Listener	Sends REL to PS via TCP connection
7	Send REL Failed	Becomes PS
8	REL ACK received	Switches channels
9	ANN received from new CS	Becomes PS
10	Timeout	Becomes CS again
11	REL received from CS	Replies with ACK, and becomes Listener
12	Button Pressed	Sends REQ to CS via TCP connection
13	REQ received from PS	Sends ACK, becomes PS
14	Send REQ Failed / Timeout	Stays as Listener
15	REQ ACK received from CS	Becomes CS

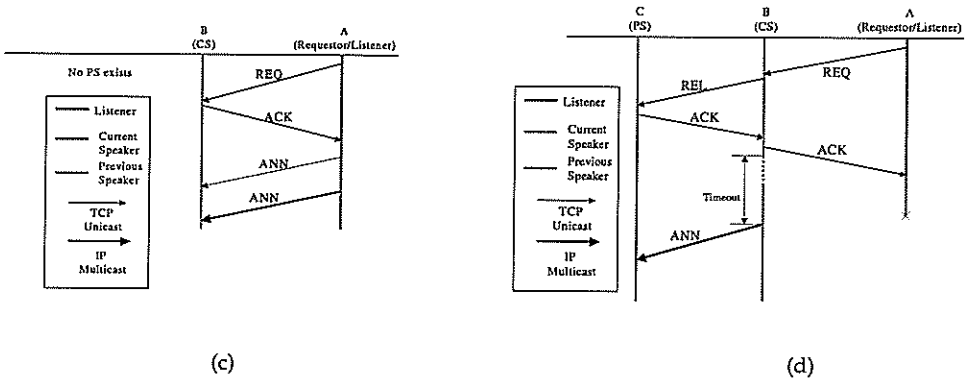
Figure 4 shows the time-line diagrams of such a protocol under different situations. (a) shows a normal round of message exchange during a speaker switching; (b) shows a message loss and retransmission; (c) shows that the same mechanism works in the simple case of two participants only; (d) shows how the previous speaker can take over as the current speaker when the current speaker is not reachable.



(a)



(b)



(c) (d)

Figure 4. Time-Line Diagram for Senatorial Algorithm

(a) Normal Operation, (b) Message Loss, (c) Simple Case with two participants only,

(d) When Current Speaker fails, Previous Speaker takes over as new speaker

The following diagram shows the control messages used in this protocol.

ANNouncement	Current Speaker	Previous Speaker	Active Channel	Previous Channel
REQuest	Workstation ID			
REQ ACK	Next Speaker	Free Channel	Active Channel	
RELease	Current Speaker	Request Workstation ID		
REL ACK	Free Channel			

If we assume the maximum one-way transmission delay is τ, it takes 5τ to finish a round of speaker switching. This switching process involves at least 6 control messages. Since only the current speaker can respond to a REQ message, no new request can be made during a switch. Moreover, since a channel has to be clean before another stream can be transmitted on it in this algorithm, participants can experience a period of time with no signal (about 2τ on the CS channel, and about τ on the PS channel).

3.2. Three-channel Algorithm (Overlapped Delay Algorithm)

Figure 5 is the network diagram for a protocol based on a three-channel algorithm.

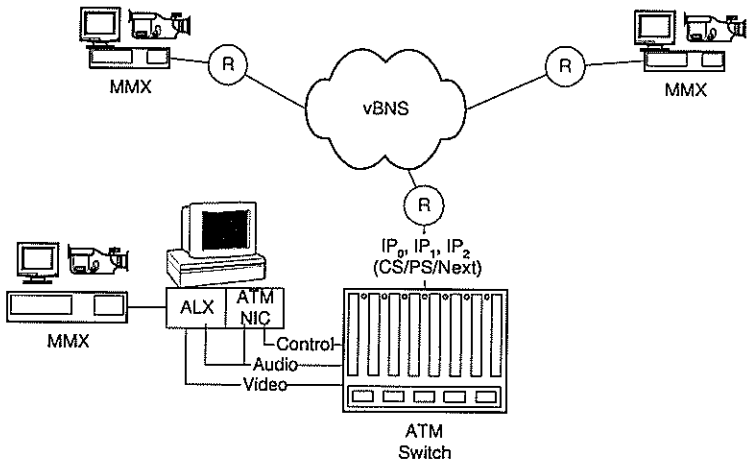


Figure 5. Overlapped Delay Algorithm

In this protocol, we use three channels instead of two. The additional channel is the "next" channel, the next available channel that a new speaker can transmit on without any conflicts. Also, these channels don't have fixed roles; instead roles rotate on these channels, as shown below. As figure 6 shows, the current channel is IP_0, the previous channel is IP_2, and the next channel is IP_1. At the time of a switch, the new speaker's transmission can be started on the next channel (IP_1) immediately after his request is acknowledged. At the same time, the next channel is constantly monitored. If a transmission is detected on that channel, the protocol can switch to that channel before receiving a notification of this speaker change first. The goal of this rotation is to eliminate the need for clearing the channel before transmission. This leads to shorter switching delay.

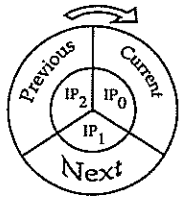


Figure 6. Previous/Current/Next Channel Rotation

In this protocol, a participant sends a request message to the current speaker and waits for the current speaker's acknowledgement. When the current speaker receives a request, he sends back an ACK message, and keeps transmitting on the same channel. When the requestor gets the acknowledgement, he starts his transmission on the next channel, and begins to periodically broadcast a SWITCH message with the speaker-switching information to all participants. A listener responds with a SWITCH ACK message when he receives a SWITCH message. The requestor keeps track of these SWITCH ACK messages. When he finds that all participants have acknowledged his SWITCH message, he becomes the new current speaker, and is able to take new requests from other participants.

To avoid blank screen caused by unsuccessful startup of the new speaker, the current speaker keeps monitoring the channel that the new speaker transmits on. If he hears from the new speaker, he just keeps his transmission on the same channel, and becomes the new previous speaker. Otherwise, if he does not detect any transmission for a period of time, he sends out a new SWITCH message with himself as the new current speaker, and starts transmitting on the new next channel. As the previous speaker, he keeps transmitting on his own channel until a new SWITCH message from a new current speaker is received. Upon receipt of this message, he stops transmission and becomes a listener.

In the case of the current speaker drop-off, the previous speaker broadcasts a SWITCH message, and starts collecting SWITCH ACK without sending REQ first. Figure 7 shows the state transition diagram of the protocol based on this algorithm.

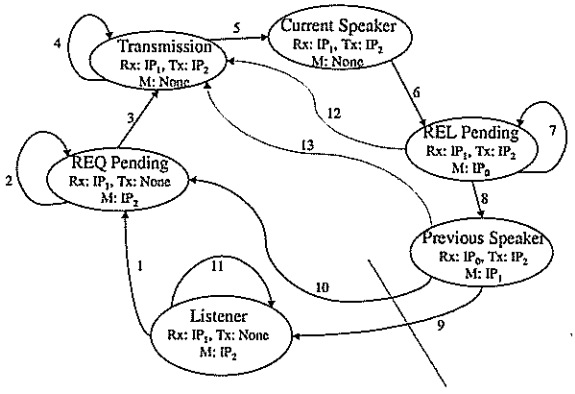
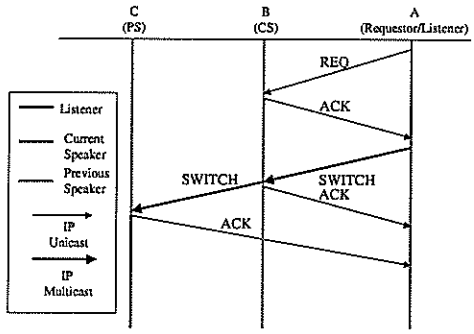


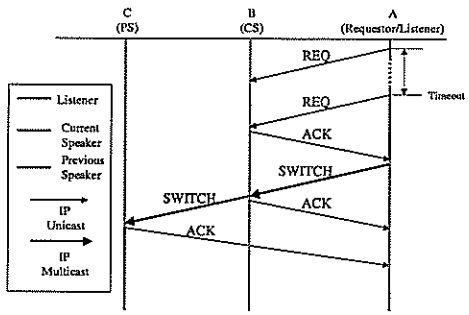
Figure 7. State Transition Diagram for Overlapped Delay Algorithm

Number	Events	Actions
1	Button Pressed	Sends REQ to CS
2	Timeout	Sends REQ to CS
3	Received ACK from CS	Sends SWITCH to all
4	Timeout	Sends SWITCH to all
5	Received ACK from all participants in list	Becomes CS
6	Received REQ	Sends ACK
7	Received SWITCH	Sends ACK
8	Received SWITCH from the new speaker	Becomes PS, and sends ACK
9	Received SWITCH	Sends ACK, and becomes Listener
10	Button Pressed	Sends REQ to CS
11	Received SWITCH	Sends ACK, and switches channels
12	Timeout, or Requestor drops off	Sends SWITCH to all
13	CS drops off (QUIT/timeout)	Tries to become CS

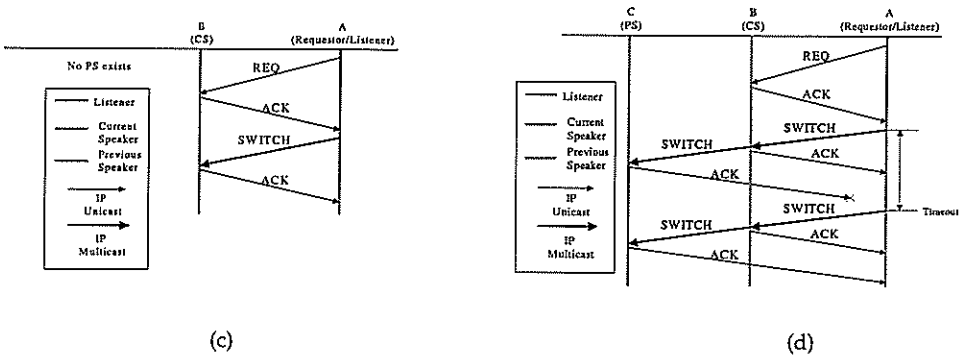
Figure 7 shows the time-line diagrams of such a protocol under different situations. (a) shows normal operations, (b) shows a message loss situation, (c) shows the simplest case of a two-participant conference, (d) shows that periodical SWITCH message broadcast can handle message loss.



(a)



(b)



(c) (d)

Figure 8. Time-Line Diagram of Overlapped Delay Algorithm

(a) Normal Operation, (b) REQ loss, (c) Two Participants case, (d) SWITCH/SWITCH ACK loss.

In this protocol, it takes 3τ to finishes a round of speaker switching. This switching process involves at least 3+n control messages, where n is the number of conference participants. Since only the current speaker can respond to a REQ message, no new request can be made before the new speaker receives acknowledgement messages from all other participants. However, because listeners monitor transmission on the next channel, the probability of a listener tuning to a blank channel is very low.

Another concern with this protocol is participant list maintenance. A requestor needs to collect acknowledgements for his SWITCH message from all participants before he can become the new current speaker. So, we need to keep a current participant list at each participant location. We can use the periodic SWITCH message to keep all participants updated with the current list. The current speaker broadcasts a SWITCH message with a participant list regularly. He also updates his list upon receipt of an ACK message. If a participant does not respond for a period of time, that participant is deemed unreachable and dropped from the list. A new participant can also be added, and the change of participant list can be made known to all participants in a one-way trip delay.

Because of the need to keep participant lists, the control messages will be long. So, although fewer control messages are exchanged, their size may lead to longer delay in this protocol. Also, the choice of parameters such as timeout has more impact on performance. The following diagram shows the control messages used in this protocol.

REQ	Current Speaker	Requestor Workstation ID	Requested Channel		
REQ ACK	Current Speaker	Active channel	Next Channel		
SWITCH	Current Speaker	Previous Speaker	Active Channel	Previous Channel	Participant List
SWITCH ACK	Current Speaker	Previous Speaker	Active Channel	Previous Channel	

3.3. Multiple-channel Algorithm

Instead of using a fixed number of channels, another distributed algorithm assigns one separate channel for each participant. Every participant always transmits on his own assigned channel

without worrying about interference with other participants. Figure 9 is the network diagram for a protocol based on this multiple-channel algorithm.

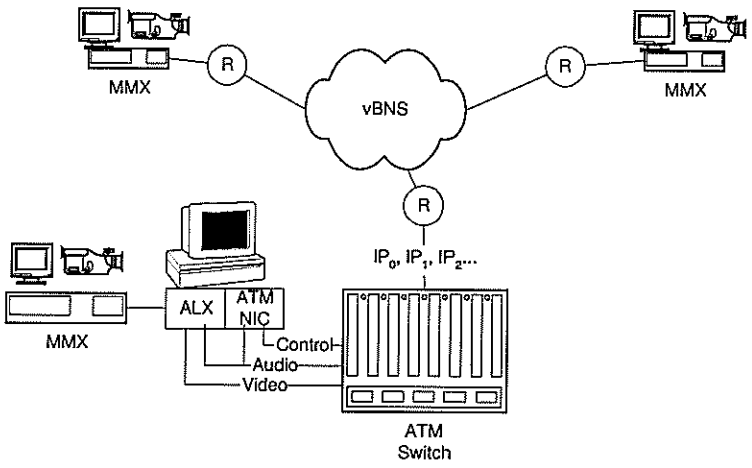


Figure 9. Multiple Channel Algorithm

Initially, all participants start as listeners. If a participant wants to speak, he broadcasts an announcement to all participants. Because each participant uses his own channel for transmission, a participant can always start transmission at the same time as he sends out an announcement.

Although there is no conflict in transmission, race conditions are possible in this protocol. Suppose that A and B both want to start transmission. Two other participants C and D may have different distances to A and B. Suppose A is closer to C, and B is closer to D. So, an announcement from A may reach C first, and an announcement from B may reach D first. C and D thus tune to A and B as the current speaker, respectively. In this case, the conference is in an inconsistent state in which participants may potentially all tune to different speakers. Similarly, the announcement from A reaches D later, and the announcement from B reaches C later. If C and D change speakers again, another inconsistency occurs.

These possible race conditions can be resolved with an additional sequence number in request messages. This idea is similar to the logical clock in a distributed mutual exclusion problem. We add a sequence number at each participant. Initially, all participants start with a sequence number of 1. When a participant decides to start his transmission, he increments his local sequence number by 1. We also use participant's workstation ID to prioritize requests. In all announcement messages, a participant adds a tuple of (sequence number, ID) to that announcement. Whenever a participant receives an announcement, he compares the sequence number in the announcement against his local sequence number. If the announcement has a higher sequence number, he tunes to the participant from whom he receives the announcement as the current speaker, and updates his local sequence number to the sequence number in the announcement. If the announcement has a lower sequence number, he just ignores it. If the announcement has the same sequence number as the local sequence number, the workstation ID of the announcer is then compared with the local record of the current speaker's workstation ID. This breaks the tie, and a speaker with higher priority ID is selected as the current speaker. As a high priority announcement proceeds, the speakers with lower priority will learn about the speaker of higher priority, stop their transmissions, and tune to this current speaker. This scheme favors more recent speakers.

Because there is no interaction among speakers, a protocol based on this algorithm is much simpler. A listener can become the current speaker by starting his own transmission and broadcasting an announcement message. Once a current speaker receives an announcement from another participant with a higher sequence number, he becomes the previous speaker. If the received announcement has the same sequence, this indicates a race condition. Depending on the workstation ID, he either stops his transmission and tunes to the new speaker, or simply ignores it. The previous speaker stops his transmission when receiving an announcement with higher priority. A listener tunes to the most recent current speaker according to the announcement messages he receives. Figure 10 shows the state transition diagram of the protocol based on this algorithm.

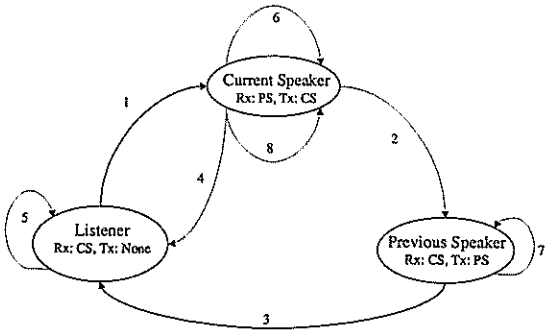
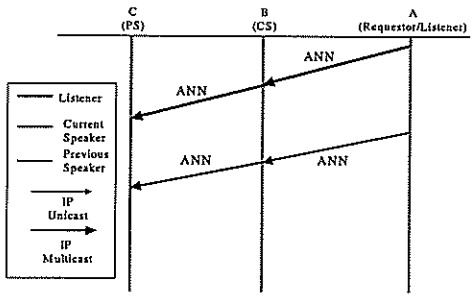


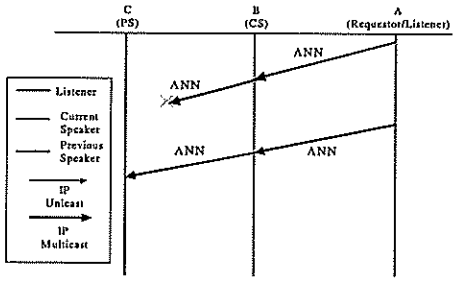
Figure 10. State Transition Diagram for Multiple Channel Algorithm

Number	Events	Actions
1	Button Pressed	Sends ANN with (seq#, station ID) to all, and starts transmission
2	Received ANN with higher priority	Switches channels, becomes PS, and adjusts local seq#
3	Received ANN with higher priority	Switches channels, becomes Listener, and adjusts local seq#
4	Received ANN with same seq# but higher priority station ID	Stops transmission, becomes Listener, and adjusts local seq#
5	Received ANN	Increments local seq#, adjusts local seq#, switch channel if ANN has higher seq# or same seq# with higher priority station ID
6	Received low priority ANN	Adjusts local seq#
7	Received higher priority ANN	Adjusts local seq#, and tunes to new CS
8	Clock ticks	Sends ANN with (seq#, station ID) to all

Figure 11 shows the time-line diagrams of such a protocol under different situations. (a) shows the normal operations, (b) shows the message loss situation, (c) and (d) show two different case of race conditions.



(a)



(b)

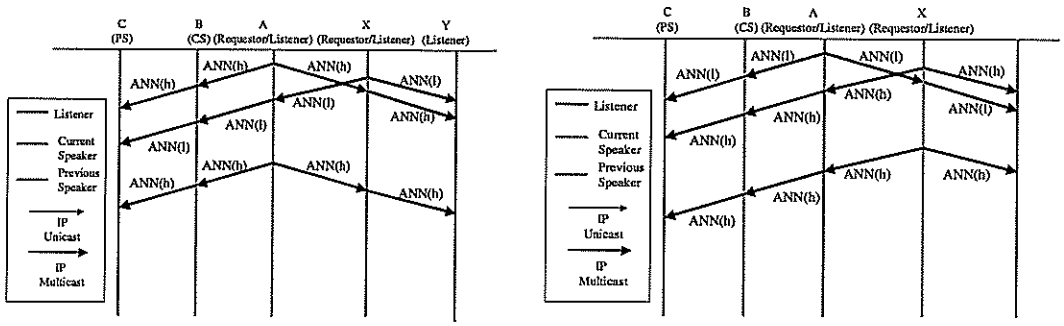


Figure 11. Time-Line Diagram of Multiple Channel Algorithm
(a) Normal Operation, (b) Message Loss,
(c) (d) Race Condition Resolution (ANN messages with different priorities reach participants in different order)

The following diagram shows the control messages used in this protocol. As we can see, because of the simplicity of this algorithm, it needs the least number of control messages.

ANNouncement	SEQ#	Station ID	Active Channel	Previous Channel

This protocol involves a delay of only τ, and one control message for speaker switching. However, because of possible race conditions, it is possible for a participant to tune to a blank channel. This won't last longer than τ, though. A major drawback of this protocol is the number of channels used. As number of participants increases, number of channels that may be in use at the same time increases linearly, and may lead to scalability problems.

3.4. Comparison of distributed algorithm-based protocols

These three protocols based on distributed algorithms have different degrees of complexity and scalability. As shown in figure 12, the left end of the spectrum is the most scalable but also the most complex algorithm, while the right end is the least scalable but also the least complicated algorithm.

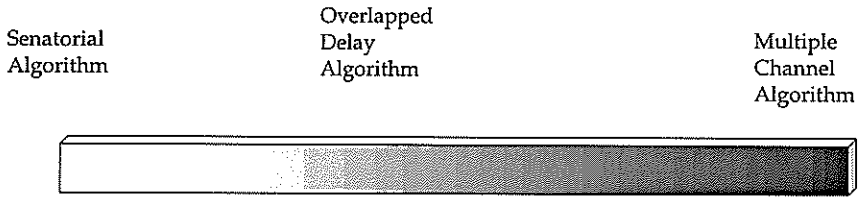


Figure 12. Comparison of Three Distributed Algorithms
(Far left end: fewer channels, more complicated control, and longer switching delay;
far right end: more channels, simpler control, and shorter switching delay)

- Senatorial Algorithm

The senatorial algorithm uses only two channels. Therefore, it is easy to scale to larger conferences. However, this algorithm requires complicated control and long switch time during which nobody transmits as a speaker.

- Overlapped Delay Algorithm

The overlapped delay algorithm uses three-channel rotation. Since the number of channels is fixed, it is easier to scale to larger conferences. Its control complexity is moderate, but with quicker switching. One drawback is that this algorithm relies on monitoring on the next channel, which may involve an additional ATM interface card, and a more complicated ALX driver.

- Multiple Channel Algorithm

This algorithm has the simplest control scheme. It is good for small conference environments. However, it uses many channels that make scaling a resource-consuming task.

4. A Protocol based on Centralized Algorithm

Figure 13 is the network diagram for a protocol based on a centralized three-channel algorithm. As its name implies, there is a centralized conference controller in this protocol. All conference control information, such as speaker changes, has to go through the conference controller.

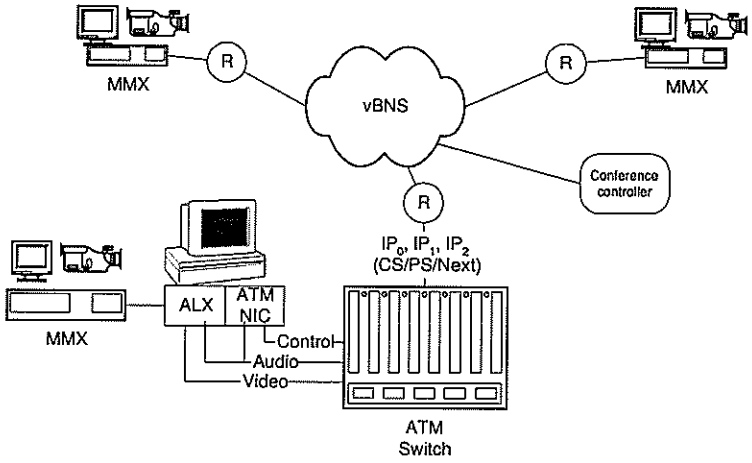


Figure 13. Centralized Three-Channel Algorithm

The three-channel rotation scheme used in this protocol is the same as in the distributed three-channel algorithm discussed before, while the other control mechanisms differ.

The conference controller keeps broadcasting conference information (INFO) to all conference participants periodically. At the same time, the conference controller also collects acknowledgements for this INFO message, and updates the participant list along with other conference information for further INFO messages. When a participant wants to speak to the conference, he sends a request to the conference controller. When the conference controller receives this request, he sends back a GRANT message that grants the requestor the floor. Then the conference controller waits for acknowledgement of this GRANT from the requestor, and does not take any further requests from other requestors.

Once the GRANT is acknowledged, the conference controller broadcasts a new INFO message with the change of speaker, and waits for acknowledgement of the new INFO message from all participants. The conference controller does not take any new requests until every participant sends an acknowledgement back. The receipt of acknowledgements from all participants indicates that all participants have synchronized to a consistent state.

In addition, the conference controller also handles participant join and drop-off. This is done using the same mechanism as in the distributed three-channel algorithm. Under some circumstances such as startup or an unreachable current speaker, the conference controller picks a new participant as the current speaker. Theoretically, the conference controller can assign any participant as the current speaker, but the assignment may depend on conference policies.

A conference participant can be in an active or inactive state. An active participant can become inactive when he drops off from the conference. An active participant can be either the current speaker, the previous speaker, or a listener. When a listener wants to speak, he sends a request to the conference controller, and expects a GRANT message back. When he receives the GRANT, he becomes the new current speaker, and starts his transmission on the next channel. The previous speaker can also request to be the current speaker with the same procedure. The current speaker becomes the previous speaker when he receives an INFO message with a new current speaker specified. The previous speaker in turn becomes a listener and stops his transmission when he receives an INFO message with a changed current speaker. However, when the previous speaker gets an INFO message that assigns himself as the current speaker, he just becomes the new current speaker with his transmission channel switched to the next channel. Additionally, all active conference participants respond to every INFO message with an acknowledgement. These acknowledgements are used to update the conference controller's participant list.

Figure 15 shows the state transition diagram of such a protocol.

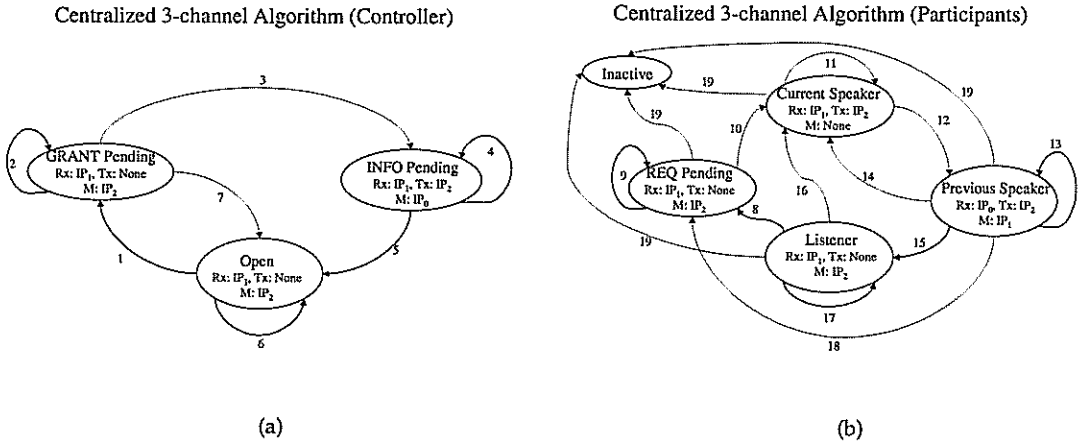


Figure 15. State Transition Diagram for Centralized Three-Channel Algorithm

(a) Centralized Conference Controller, (b) Participant.

Number	Events	Actions
1	Received REQ	Sends GRANT to Requestor
2	Timeout	Sends GRANT to Requestor
3	Received ACK from Requestor	CS changes, sends INFO to all
4	Timeout	Sends INFO to all
5	Received INFO ACK from all	Ready to accept REQ
6	Timeout, receives INFO ACK, receives QUIT	Sends INFO to all, and updates participants list
7	Requestor lost contact	Ready to accept REQ
8	Button Pressed	Sends REQ to CC
9	Timeout	Sends REQ to CC
10	Received GRANT from CC	Sends INFO to all, and becomes CS
11	Received INFO	Replies INFO ACK
12	Received INFO with role change	Becomes PS
13	Received INFO	Sends INFO ACK
14	Received INFO with role change	Becomes CS, and sends INFO ACK
15	Received INFO with role change	Becomes Listener, and sends ACK
16	Received INFO with role change	Sends INFO ACK, and becomes CS
17	Received INFO	Sends INFO ACK
18	Button Pressed, or received INFO	Sends REQ to CC
19	QUIT button pressed	Sends QUIT message to CC

Figure 16 shows the time-line diagrams of such a protocol under different situations. (a) shows the normal operations, (b) shows cases of message losses, (c) shows a simple case of a two-participant conference.

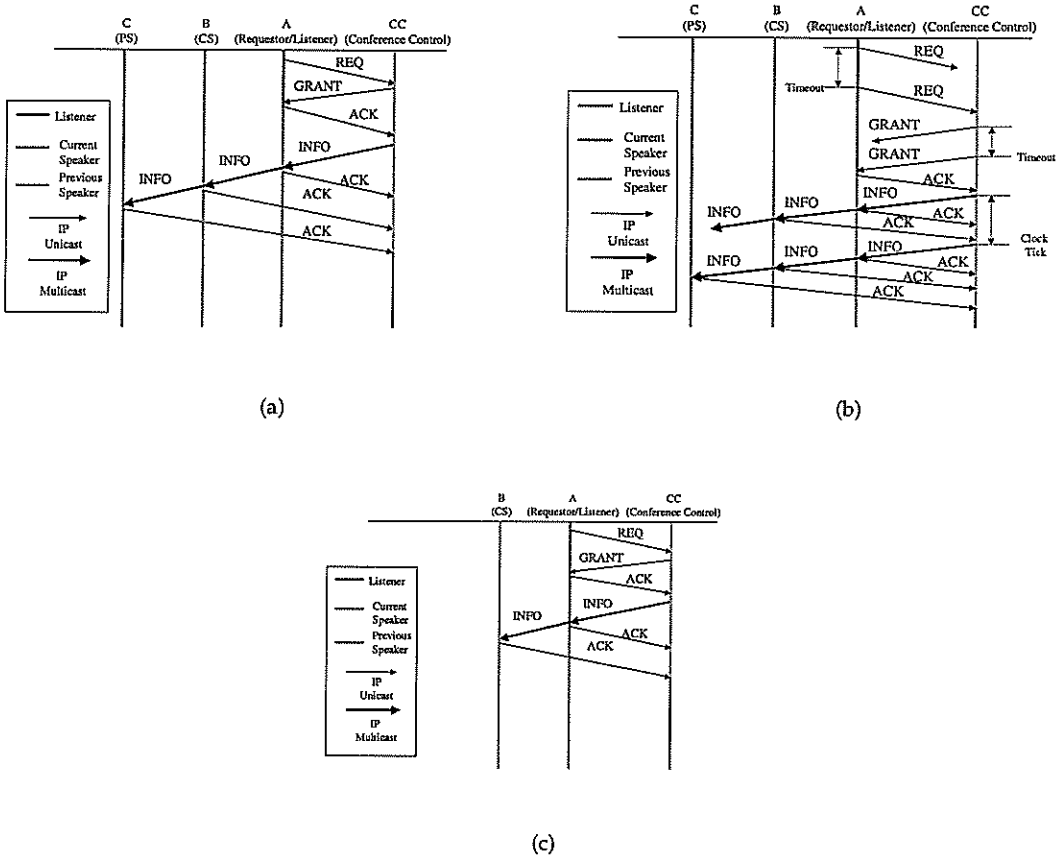


Figure 14. Time-Line Diagram of Centralized Three-Channel Algorithm

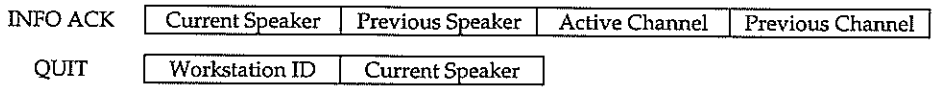
(a) Normal Operation, (b)Message Loss, (c) Two Participants Only Case.

In this protocol, it takes 4τ of time to finish a round of speaker switching. This switching process involves at least 4+n control messages, where n is the number of conference participants.

As a protocol based on a centralized algorithm, this protocol avoids possible inconsistent states in a distributed algorithm based protocol. However, because of the introduction of the centralized conference controller, a single point of failure problem is also introduced. This can be solved with additional backup conference controllers with different sets of three channels at different sites. But this approach clearly takes more channel resources and restricts conference scalability.

The following diagram shows the control messages used in this protocol.

REQ	Current Speaker	Requestor Workstation ID	Requested Channel		
GRANT	Previous Speaker	Active Channel	Next Channel		
GRANT ACK	Current Speaker	Previous Speaker	Active Channel	Previous Channel	
INFO	Current Speaker	Previous Channel	Active Channel	Previous Channel	Participant List



5. Discussion and Future Work

As we can see, protocols based on distributed algorithms have shorter delays caused by message exchanges, but they share complicated control mechanisms and tricky failure recovery problems. In contrast, protocols based on centralized algorithms are generally simpler and easier to recover from failures, but they have generally longer message delay and are subject to single-point-of–failure problems.

In the future, we plan to implement the centralized protocol for initial local simulation and test. For comparison, all three other distributed protocols will also be implemented and simulated locally. We will monitor normal operation of these protocols as well as operations under failures. We will also measure switching delays and control message volume. Another task is to experiment with all parameters in these protocols, such as the interval between control messages, the buffer flushing interval, etc. Once we have this information in hand, we will evaluate these protocols. When the ALX driver is ready, we will test our protocol in a real vBNS environment.

6. Summary

The ALX has been designed to extend MMX's video-conferencing to wider area vBNS. We cover the floor control protocol to be used in ALX video-conference applications and show the requirements for such protocols. We discuss three protocols based on distributed algorithms and one protocol based on a centralized algorithm and list advantages and drawbacks of each protocol. A protocol based on a centralized algorithm is our current choice for implementation. However, we need obtain our simulation results to make the final decision.

Bibliography:

[1] Jamison, J., Wilder, R., "vBNS: The Internet Fast Lane for Research and Education", *IEEE Communications Magazine*, January 1997.

[2] Parwatikar, J., Vaudeville, http://www.cs.wustl.edu/cs/playground/vaudeville/.

[3] Goldman, K., Swaminathan, B., McCartney, T., Anderson, M., Sethuraman, R., "The Programmers' Playground: I/O Abstraction for User-Configurable Distributed Applications." *IEEE Transactions on Software Engineering*, September 1995.

[4] Programmer's Playground, http://www.cs.wustl.edu/cs/playground/.

[5] Malpani, R., Rowe, L., "Floor Control for Large-Scale MBone Seminars", Proceeding of ACM Multimedia, Seattle, WA, 1997.

[6] Handley, M., Wakeman, I., and Crowcroft, J., "The Conference Control Channel Protocol (CCCP): A Scalable Base for Building Conference Control Applications", ACM SIGCOMM 95, New York, August 1995.

[7] Bormann, C., Ott, J., Reichert, C., "SCCP: Simple Conference Control Protocol", Internet Draft draft-ietf-mmusic-sccp-00.txt, Work in Progress, June 1996.

[8] T.120 Standards for Audiographic Teleconferencing, ITU.