AVC-820

Telecommunication Standardization
Sector
Original: English
(TSS)

Experts Group for Video Coding and Systems in ATM and Other Network Environments **August 21, 1995**

STUDY GROUP 15
CONTRIBUTION

Source: InSoft, Incorporated
email: rhp@insoft.com
voice: +1 (717) 730-9501
fax: +1 (717) 703-9504

Title: Non-MCU Multipoint Call Setup

Date: 8/21/95

I. Overview

Non-MCU Multipoint LAN conferences is a standard feature of current LAN-based video conferencing systems. For H.323 gateways and terminals to be commercially viable products, they must support multipoint calls. Customers will expect that terminals and gateways will be able to participate in these multipoint calls. The result of not addressing this requirement is that each gateway and terminal will by necessity have a unique, proprietary method of multipoint setup.

It follows that the initial H.323 specification must include LAN based multipoint call setup. Unless the multipoint session setup is specified, there will be no guidance given to gateway and terminal providers who wish to deliver interoperable products meeting this customer requirement. The remainder of this paper outlines a the structure of such a protocol.

In a non-MCU multipoint conference, there is no central mediator. Every endpoint maintains the state of the conference, and keeps a full mesh of connections to other conference members. This is accomplished via a predefined network protocol and a finite state machine at each endpoint.

II. H.323 Conference Engine

An H.323 Conference Engine is the core component on each endpoint which holds the conference state. One Conference Engine runs per user. On multi-user systems, more than one Conference Engine may run on a single CPU. The Conference Engine provides these services over a virtual network, layered over actual networks, giving user applications true interoperability.

III. Endpoint Applications (Plugins)

The conference engines at each endpoint can support a number of distributed applications which will be referred to as Plugins to this state machine. These Plugins can be any number of distributed applications that can utilize the services of the conference engine at that endpoint.

IV. States

The following details the states of the conference engine:

1. ***Dead***
   No conferencing activity is currently in progress for this node.

2. ***Prompting***
   Node received invitation and is waiting for user to decide whether to accept

3. ***Accepted***

Node *Accepted*, waiting for inviter to complete the handshake

4. ***Running***
   Node is active in the conference. In the *Running* state, there is a roster of conference members, each of which is marked either *Active* (has *Accepted*, and can be communicated to in the session) or *Pending* (has not yet *Accepted*).

V. State Transitions

The following explanatory text details the state transitions of an H.323 conference engine.

- ***Dead to Prompting***
  When a member in the *Dead* state receives an invitation, it transitions to *Prompting* and waits for the user to accept or reject this invitation.

- ***Prompted to Dead***
  If the user rejects, the system sends a rejection, drops the connection, and returns to *Dead.*

- ***Prompted to Accepted***
  If the user accepts an invitation request, the system tells the inviter of this state transition, and transitions the local node to *Accepted.*

- ***Accepted to Running***
  After the prompted user accepts the invitation, the inviter will then send a roster of conference members to the invited system. Next, the invited system will transition to *Running* and connect to all of the currently active members. When a *Pending* member connects, the member receiving the connection will mark this conference member as *Active.*

- ***Dead to Running (Conference Initiator)***
  To initiate a conference, the system transitions from *Dead* state to the *Running* state. This endpoint connects to each one of the requested conference members. Next it marks each of these new endoints as *Pending* and sends each of the new endpoints an invitation. When a member accepts, this members entry is marked as *Active*, and the initiator sends a roster of conference members to the newly *Active* member. If the endpoint rejects, (or if the connection drops), it is dropped from the roster, and all *Active* members are notified.

- ***Member Addition***
  To add a member to an existing conference, the initiating system adds this member to the member list and marks this entry as *Pending.* Next, the inviter connects to the new system and sends it an invitation. Following this invitation, the inviting system notifies all Active members that a new

Pending member has been added. The acceptance of rejection of this new system is then treated as detailed above.

- ***Simultaneous State Update***
  Corruption caused by simultaneous state update is prevented through the use of mutex locks. These mutex locks are implemented via a token ring scheme between all active conference participants.

VI. Endpoint Actions

Each endpoint will need to have a set of actions to keep the distributed state machine current. The following is a full list of these actions:

- **Initiate(S)**
  S is set of prospective members. This is valid to perform when the state of the engine is *Dead*. Initiate will first establish network connection to each member of S. Next, it will send each one Identity and Invitation and then transition to *Running*({},S). For any member which could not be reached, that node is marked as if it rejected the invitation with the memberRejected event.

- **Accept(I)**
  This is valid to perform when the current state is *Prompting*(I). Acceptance is sent to I and the state of the engine at this node is transitioned to *Accepted*(I).

- **Reject(I,R)**
  This is valid to perform when the current state of the engine is *Prompting*(I). R is a reason for the rejection. When this action is performed, a message is sent to I and the local state machine is transitioned to the *Dead* state.

- **Leave()**
  This is valid to perform when state is *Running*(A,P). A Farewell messages is sent to all members of A and P then all opened connections are closed. Next, the local state machine is transitioned to the *Dead* state. (Note that it is actually possible to perform Leave if the state is *Accepted*(I) or *Prompting*(I), in which case Leave is equivalent to Reject(I,Rejected).)

- **Add(M)**
  M is new member to add to the conference session. This is valid to perform when the state is *Running*(A,P). First, a network connection is established to M and it is sent an Identity and Invitation. Next, the state machine is transitioned to *Running*(A,P+{M}). If M could not be contacted, that node is marked as if it rejected the invitation with the memberRejected event.

Notes for Initiate and Add
A member to contact is specified by a triple {Username, Hostname, Addresses}. <Username>@<Hostname> is the user's written address. Addresses is a list of network addresses for the user's machine (a list is required since machines are capable of supporting multiple network interfaces). The addresses are listed in the order in which they are preferred. (i.e., try address 1 first, then address 2, etc. If the list is empty, Hostname is used to look up one address A; Addresses is then the 1-entry list {A}.) Once a network-level connection has been made, the inviter sends a ConnectName(Username) message to indicate which user is being invited. (The same procedure is done when a new member receives a Greeting message and needs to connect to the currently active members.)

VII. Messages

In order to carry out the above actions, the following set of messages must be employed:

- **Invitation (List of prospective members)**
  This message is valid to receive when current state is *Dead*. This Invitation is then relayed to the conference manager/node controller application. The engine is then transitioned to *Prompting*. When received at inappropriate time, a rejection, Rejection(LineBusy), is sent to the inviter and there is no further state transition.

- **Acceptance (No arguments)**
  This message is valid to receive when the state is *Running*(A,P), and the sender is in P. When this messages is received, the node should reply with a Greeting(A'+{self},P') and transition to *Running*(A',P'), where A'=A+{sender} and P'=P-{sender}.

- **Rejection(R)**
  This message is valid to receive when the state is *Running*(A,P), and the sender is in P. R is the reason the invitation was rejected. When received, the node should transition to *Running*(A,P-{sender}) or to *Dead* if A and P-{sender} are both empty.

- **Greeting(A,P)**
  This messages is valid to receive when state is *Accepted*. This can only be sent when state is *Running*(A,P). When received, the node should connect to all members in A (except for itself and the sender), and send each one Identity and Activate. Next, the node should transition to *Running*(A-{self},P). If any of these connections fail, Leave the conference immediately (otherwise, a non-fully connected conference could result which would cause serious inconsistencies).

- **Farewell (No arguments)**

This messages is valid to receive at any time. If the current state is *Running*(A,P), and sender is in A, the node should transition to *Running*(A-{sender},P). If sender is in P, treat Farewell as equivalent to Rejection(Unknown). If the state is *Accepted*(I) or *Prompting*(I), and sender is same as I, transition to *Dead*. If state is now *Dead*, the connection should then be closed.

⟨ **Identity**
This message is valid to receive at any time. Included in the message is the address list, userName, and hostName. Record this information, associated with sender's network connection, but don't transition.

⟨ **Activate(H)**
This messages is valid to receive when state is *Running*(A,P) and H is in P. The H in this messages is the unique user handle (an identifier for the conference session). This message is sent by a member who has just connected, after having been invited by someone else. On receipt, move H from P to A and associate the connection with userHandle H.

⟨ **Pseudo-Message: Connection**
When anyone connects, an Identity message should be sent immediately to the connecting node.

⟨ **Pseudo-Message: Connection died**
A network connection has been broken from a remote end. In state *Running*(A,P), if sender is in P, treat this as if a Rejection(Unknown) was received, and this endpoint should transition to *Running*(A,P-{sender}). Otherwise, the broken connection should be treated as if a Farewell was received, and transition to *Running*(A-{sender},P). If the current state is *Accepted*(I) or *Prompting*(I) and if this local node was the sender, it should transition to *Dead*.

⟨ **ConnectName(U)**
This messages is sent from A to B when A first connects to B. U is a NULL-terminated string, the username of the user being invited. On a multi-user system, this allows multiple users to run at the same time. Incoming connections can be directed to the appropriate conference engine based on their ConnectName. On a single-user system, this serves as a sanity check, to make sure that the user being invited is the one currently using the system.

Data(Arguments not all listed here)
When a data packet is received, route it to the appropriate endpoint application (plugin). If the appropriate plugin is not available, reply with a bounce packet (if supported: *See Feature Flags*).

VIII. Feature Flags

This protocol includes some optional features with the capability to be expanded as needed. As new capabilities are added to the engine, a new flag is defined for each. These flags indicate the set of capabilities which an engine implements and are exchanged in the Identity message, in appCaps.featureFlags. Currently defined features include:

- ***Ping***

  An engine can give a bounce message when a packet is received for an inactive plugin. This bounce message is in the Ping protocol:

  1. send a Ping to a plugin
  2. if it is registered, a PingReply is sent;
  3. if it is not registered, a bounce message is returned

  If a Ping is addressed to a member whose engine does not have the Ping feature, the engine sends a CantPing to the originator instead of forwarding the Ping.

- ***Clip***

  Includes a clipboard facility, which can be accessed remotely via the network connection. If a clipboard request is addressed to a member whose engine does not have the Clip feature, the engine sends a clipboard NoClip message to the originator instead of forwarding the request.

- ***Mutex***

  Supports mutex locks in the conference. An engine which does not have the Mutex feature is left out of the token ring used for mutex locks.

- ***Reasons***

  Includes a reason argument in the Rejection message. A Rejection message received from an engine which does not include this flag is treated as if the reason were Invalid.

These flags allow backwards compatibility with older conference engines. They also allow engines to behave properly when talking to a new engine with some features not yet implemented. This allows developers to test their engines against a reference implementation early in the development cycle, before implementing the complete feature set.

VIIII. Protocol Specification

The following is the protocol specification for the H.323 conference engine.

H.323 Conference Engine PDU Header:

Description	Octets	Notes

PDU length	1-4	
Protocol major version number	5-6	Currently 2
Protocol minor version number	7-8	Currently 0
PDU type tag	9	See ConfdMsgTag
Alignment bytes	10-12	Reserved; set to 0
PDU Argument, if any	13	See below.

There is a different PDU Argument format for each type of message. Some messages have no argument.

Message: *Invitation (type tag confdMsgTagInvitation, which is 1)*

Description	**Octets**	**Notes**
inviterIndex	1	Index of sender's entry in roster
Number of members (N)	2	Number of members in roster
Alignment	3-4	Reserved; set to 0
Roster. N entries; format of each is DveConfParticipant (q.v.)		
Entry 0	5-124	
Entry 1	125-244	
Entry (N-1)	5+120*(N-1) to 5+120*N	

Message: *Acceptance (type tag confdMsgTagAcceptance, which is 2)*

Acceptance has no argument.

Message: *Rejection (type tag confdMsgTagRejection, which is 3)*

Description	**Octets**	**Notes**
reasonCode	1-4	Rejection reason. A reason code is a ConfdRejectionReason (q.v.)

Message: *Greeting (type tag confdMsgTagGreeting, which is 4)*

Description	**Octets**	**Notes**
Number of members (N)	1	
Recipient index	2	Index of recipients entry in the roster (below)
Alignment	3-4	Reserved; set to 0
N roster entries	5-(4+N*size)	
Entry format (relative to start of entry):		
Member	1-s (q.v.)	DveConfParticipantLong
Active flags	+1-s+2	Boolean
Alignments	+2-s+3	Reserved; set to 0

Message: *Farewell (type tag confdMsgTagFarewell, which is 5)*

Farewell has no argument.

Message: *Identity (type tag confdMsgTagIdentity, which is 6)*

Description	Octets	Notes
addresses	1-452	DvelPAList
appCaps	453-472	ConfdAppCaps
serial	473-536	0-terminated str, ""
userName	537-569	0-terminated str, username
hostName	570-634	0-terminated str, hostname

Message: *ConnectName (type tag confdMsgTagConnectName, which is 7)*

Description	Octets	Notes
userName	1-33	

Message: *PluginData (type tag confdMsgTagPluginData, which is 8)*

Description	Octets	Notes
size	1-4	.
Protocol major version #	5-6	Currently 0
Protocol minor version #	7-8	Currently 1
Destination port	9-12	
Source port	13-16	
Routing flags	17-20	
Plugin Protocol	21-24	
Message tag	25-28	
Recipient list	29-60	DvePartList
Payload	61-	Optional.

Message: *RejectionReport (type tag confdMsgTagRejectionReport, which is 9)*

Description	Octets	Notes
Rejecter	1-120	DveConfParticipant
Reason	121-124	ConfdRejectionReason

Message: *NewMember (type tag confdMsgTagNewMember, which is 10)*

Description	Octets	Notes
Member	1-120	DveConfParticipant

Message: *Activate (type tag confdMsgTagActivate, which is 11)*

Description	Octets	Notes
User handle	1	

Data structures used above:

ConfdAppCaps:

Description	**Octets**	**Notes**
Hardware type	1-2	
OS type	3-4	
Feature flags	5-8	
App's major version #	9-10	
App's minor version #	11-12	
Max participants app supports	3-14	
Max UDP datagram size	15-16	
Real hardware type	17-18	For emulation support
Real OS type	19-20	

DveIPAList:

Description	**Octets**	**Notes**
16 address entries	1-448	
Entry format (relative to entry start):		
Host address	1-20	DveIPA
Port #	21-22	
Reserved (set to 0)	23-28	
Number of addresses	449-452	

DVEConfParticipant:

Description	**Octets**	**Notes**
Address	1-20	DveIPA
userName	21-52	0-terminated
hostName	53-116	0-terminated
userHandle	117	Handle of member in conference
Alignment	118-12	Reserved; set to 0

DveConfParticipantLong:

Description	**Octets**	**Notes**
Address list	1-	DveIPAList
Member		DveConfParticipant

ConfdRejectionReason:

The following is the list of the rejection reasons that could be encountered during the course of a conference session:

- *Not Present* (confdRejectionReasonNotPresent, which is 1)
- *Line Busy* (confdRejectionReasonLineBusy, which is 2)
- *Rejected* (confdRejectionReasonRejected, which is 3)

- *Can't Connect: Unknown Host* (confdRejectionReasonCantConnect_HostUnknown, which is 4)
- *Can't Connect: Unreachable Host* (confdRejectionReasonCantConnect_HostUnreachable, which is 5)
- *Can't Connect: Unreachable Network* (confdRejectionReasonCantConnect_NetworkUnreachable, which is 6)
- *Can't Connect: No Registrar* (confdRejectionReasonCantConnect_NoRegistrar, which is 7)
- *Can't Connect: No Engine* (confdRejectionReasonCantConnect_NoEngine, which is 8)
- *Can't Connect: Local* (confdRejectionReasonCantConnect_Local, which is 9)
- *Can't Connect: Unknown* (confdRejectionReasonCantConnect_Unknown, which is 10)
- *Can't Connect: Unknown Host* (Remote) (confdRejectionReasonRemoteCantConnect_HostUnknown, which is 10)
- *Can't Connect: Unreachable Host* (Remote) (confdRejectionReasonRemoteCantConnect_HostUnreachable, which is 11)
- *Can't Connect: Unreachable Network* (Remote) (confdRejectionReasonRemoteCantConnect_NetworkUnreachable, which is 12)
- *Can't Connect: No Registrar* (Remote) (confdRejectionReasonRemoteCantConnect_NoRegistrar, which is 13)
- *Can't Connect: No Engine* (Remote) (confdRejectionReasonRemoteCantConnect_NoEngine, which is 14)
- *Can't Connect: Local* (Remote) (confdRejectionReasonRemoteCantConnect_Local, which is 15)
- *Can't Connect: Unknown* (Remote) (confdRejectionReasonRemoteCantConnect_Unknown, which is 16)