Telecommunication Standardization Sector
Original: English
(TSS)

AVC-827 Intel-3

Experts Group for Video Coding and Systems in ATM and Other Network Environments

October 24-27, 1995

STUDY GROUP 15
CONTRIBUTION

Source: Jim Toga, Intel Corporation
email: jim_toga@ccm.jf.intel.com
voice: +1 (503) 264-8816
fax: +1 (503) 264-3485

Colin Hulme
email: colin_hulme@ccm.jf.intel.com
voice: +1 (503) 264-8022
Tom Bezdicek
email: thomas_m_bezdicek@ccm.jf.intel.com
voice: +1 (503) 264-8020
Don Evans
email: don_p_evans@ccm.jf.intel.com
voice: +1 (503) 264-8682

Title: Gatekeeper and Connection Setup in H.323

Date: October 13, 1995

1. Introduction

This proposal recommends a protocol description and operational model for utilizing a Gatekeeper entity in an H.323 environment. As specified in the H.323 draft, the Gatekeeper is described as a separate logical entity, it may be physically combined with any other endpoint described in the H.323 documents.

Sections 4 and 5 will outline procedures to be used in the call connection and setup with or without Gatekeeper involvement. Section 7 describes the PDUs that should be added to H.22Z for usage in the procedures. Section 8 contains some additional notes and options.

2. Background

It is expected that *if* Gatekeepers are present in the LAN environment, they will be utilized by any H.323 nodes (this includes but is not limited to, terminals, Gateways, and MCs). This is not meant to imply that the presence on Gatekeepers is *required* in the H.323 environment.

Gatekeepers will provide admission control to LAN based conferencing, based upon bandwidth resource usage. Either MC or native NOS security should provide admission control based upon security access. Authentication of endusers *should* occur (to the extent desired) between Gatekeepers and any terminals that register with them. It is expected that the primary authentication mechanism for security purposes with respect to the LAN, will be supplied at the terminal/application interface.

Gatekeeper to Gatekeeper cooperation is encouraged for enterprise management, however there is no requirement that Gatekeepers communicate outside of connection setup/permission exchanges. The ability to cache information for performance or efficiency issues is up to implementors of a gatekeeper.

2.1 Definitions

Network Address - this is meant to encompass all of the addressing information that is needed for end to end application communication. It includes both the physical transport address and the logical port to which the application is bound.
Guardian - this is the term given to a Gatekeeper with which a terminal(s) has bound. A terminal will attempt connections to other terminals via contacting its *guardian* gatekeeper.
Node - this term is used to denote any H.323 specified components as listed in sec 5 of H.323 Draft.
Terminal - is used as shorthand for the more formally defined H.323 terminal.

2.2 Symbols

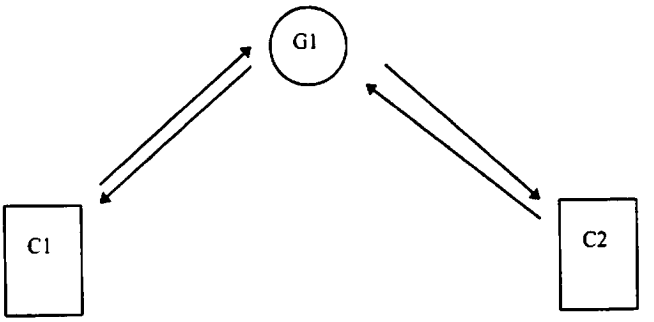
BRQ - Bandwidth Request
BCF - Bandwidth Confirmation
BRJ - Bandwidth Reject
CRQ - Connection Request
CCF - Connection Confirmation
CRJ - Connection Reject
CIP - Connection In Progress
DRQ - Disconnect Request
GRQ - Guardian Request
GCF - Guardian Confirmation
GRJ - Guardian Reject
MRQ - Management Request
MCF - Management Confirmation
MRJ - Management Reject
NLR - Node List Request
NRL - Node Response List
RRQ - Registration Request
RCF - Registration Confirmation
RRJ - Registration Reject
SRQ - Status Request
SRR - Status Report Response

3. Framework and Models

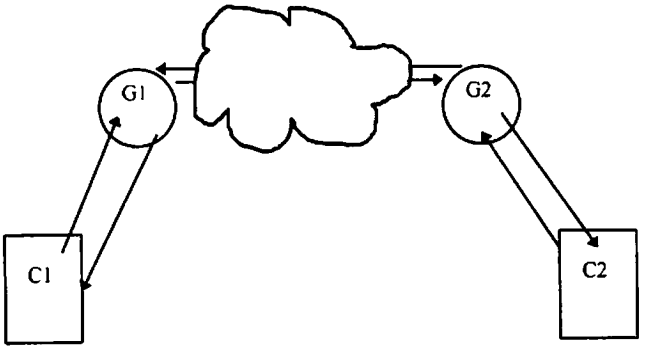
The connection model for terminals on the LAN connecting to other terminals will be the same as terminals connecting to Gateways. Stated in another manner, Gateways will appear to Gatekeepers and other H.323 terminals as an H.323 terminal.

As established in the Ptel1 submission and current H.323 draft specification, terminals will register with gatekeepers to establish a *binding* between the two. Thereafter, connection setup will be gated through this gatekeeper (and the corresponding gatekeeper at the called end). Terminals should re-establish this binding in the event that it becomes invalid. The consistent mapping of this binding is outside the scope of this proposal but some solutions will be offered here. Although initial call setup and establishment of the control channel occurs via interaction with the respective gatekeepers, continued conference exchanges may occur without gatekeeper involvement. The two exceptions to this, are the disconnection of terminal(s) from a conference and the unsolicited status (section 7).

There are two basic models of gatekeeper permissioned conferences: shared gatekeeper and peer gatekeepers. Shared gatekeepers imply that the two (or more) terminals have the same guardian. The peer gatekeeper model is present when two terminals do *not* have the same guardian. This does not imply that the peer gatekeepers must be logical or physical neighbors in a network. The two models described are shown in the respective figures that follow. The absence of a gatekeeper binding on either end of the conference shall not be disallowed by the protocol or procedures.



(Figure 1)

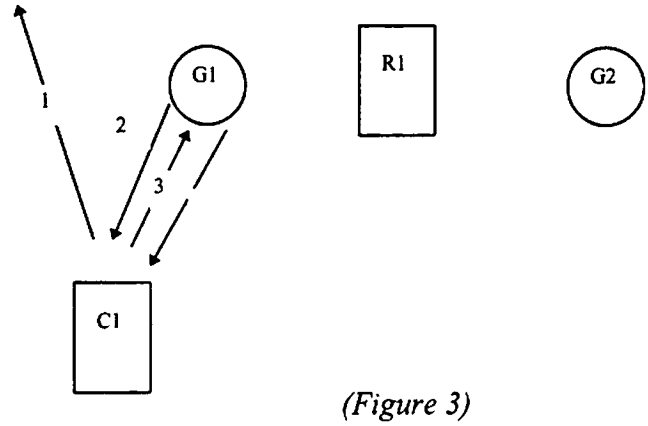


(Figure 2)

4. Terminal-Gatekeeper Registration

There are two methods in which a terminal may find and become registered/*bound* to its gatekeeper. The first method is an auto-binding mode, primarily controlled by physical topology. The second method provides for a deterministic, static binding.

4.1 Auto-Binding



(Figure 3)

C1 starts an H.323 video application. There is no gatekeeper address in the client.

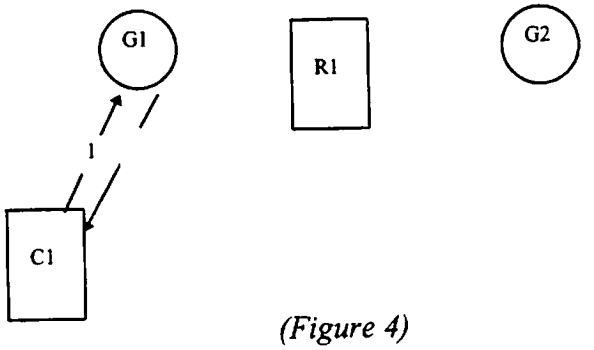
1. C1 broadcasts Guardian Request (GRQ), asking "who is my gatekeeper".
   This broadcast is targeted to a well known port.
2. Ideally router R1 is configured to block the broadcast so only gatekeeper G1 sees it and responds. In the event that router R1 doesn't block the broadcast, both gatekeepers G1 and G2 may respond with GuardianConfirmation (GCF); "I can be your gatekeeper".
   C1 takes a response and may cache that information.
3. C1 sends a Registration Request (RRQ) with a flag set indicating to "Bind" to G1.
   G1 detects the bind flag and may create a new cache entry for C1.
4. G1 sends a Registration Confirmation (RCF) back to C1.

If at any time a client determines it has an invalid binding with its guardian, it must rebind. The invalid binding may be detected by either an **RRJ** (with an **Not Bound** status) or a timeout on an **RRQ**. Before it re-binds, it should issue an **GRQ** to discover its 'owning' gatekeeper.

The terminal may issue the **RRQ** with the bind flag set to TRUE *only* if it receives a **GCF** or a timeout. (receiving only **GRJ's** constitues denial of permission to conference).

Auto-binding allows for lower administrative overhead in configuring individual H.323 terminals and additionally allows replacement of an existing gatekeeper without reconfiguring all of the affected terminals.

4.2 Static-Binding



(Figure 4)

1. C1 sends a Registration Request (RRQ) with a flag set indicating to "Bind" to G1.
   G1 detects the bind flag and creates a new registration/binding entry for C1.
2. G1 sends a Registration Confirmation (RCF) back to C1.

As stated in section 7.1 of the H.323 draft, as part of the manual configuration of a terminal, the location of its *guardian* gatekeeper may be entered.

4.3 Deterministic-Binding

The default binding behavior of the gatekeeper should be configurable (whether to respond to a **GRQ** broadcast affirmative (**GCF**) or negative (**GRJ**)).

This functionality will provide for consistent binding between a terminal and a particular gatekeeper. Additionally this may provide the ability to block a particular from conferencing. This blocking may be accomplished through the use of a negative response to GRQ (GRJ). If configured for an affirmative response, the gatekeeper will reply with a GCF. If configured for a negative response, the gatekeeper will reply GRJ.

If a terminal receives one or more negative responses and no positive responses, the terminal may not start a conference. A terminal receiving no responses, assumes tacit approval to initiate a conference. (note: there is a 'rogue' conference discovery mechanism described in Sec. 8)

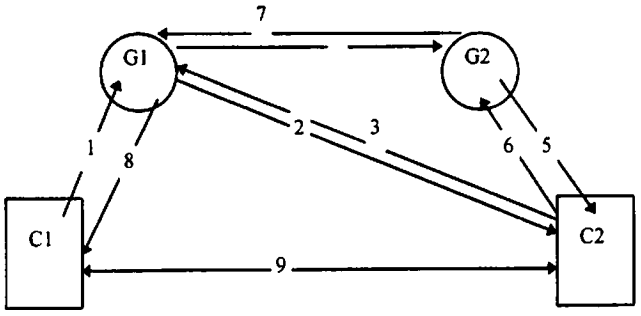
A customer that so desired might set default behavior of a gatekeeper to negative response, and add users to a affirmative response list. This would determine the binding, and guarantee that no one uses bandwidth if their gatekeeper is inactive.

Any hybrid of auto-binding and static binding can be provided with this model.

5. Call Connection/Setup

This section will outline a number of possible LAN environments and reference the PDUs that are exchanged to initiate and complete, a call between terminals; the end state being the establishment of a reliable transport connection carrying logical channel 0 (the control channel).

5.1 Both Clients bound to gatekeepers

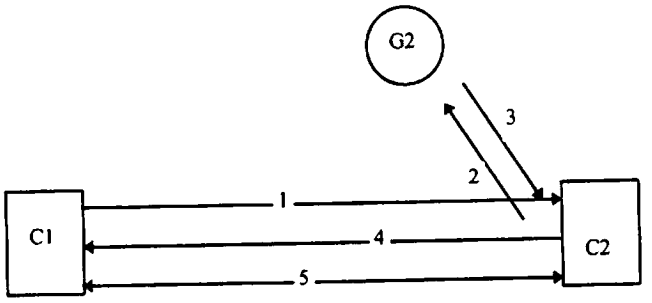


(Figure 5)

1.	CRQ for C2	(C1 sends connection request for C2)
2.	MRQ	(G1 Queries C2 for C2's gatekeeper)
3.	MCF	(G1 may cache C2-G2)
4.	CRQ for C2 from C1/G1	(G2 may cache C1-G1)
5.	CRQ	(G2 passes C1's CRQ to C2)
6.	CCF	(C2 accepts connection)
7.	CCF	
8.	CCF	
9.	*Control Channel Established*	

If C2 is already contained in G1's cache, steps 2 and 3 may be eliminated. If C2 is not contained in G1's cache, but another client on the same subnet as C2 is contained in the cache, steps 2 and 3 may be eliminated.

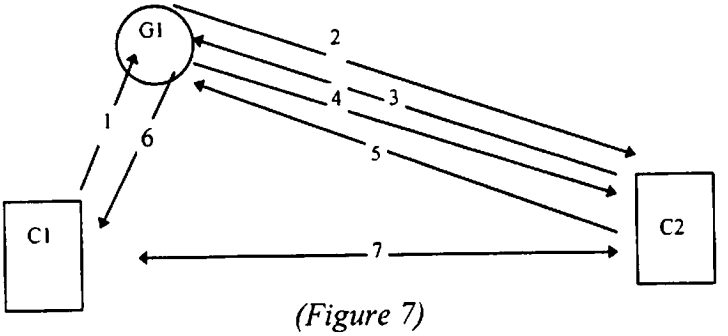
5.2 Callee bound to gatekeeper



(Figure 6)

1. CRQ for C2 from C1/?? (C1 indicates it has no gatekeeper)
2. CRQ from C1 (C2 queries G2 for approval)
3. CCF (G2 allows for bandwidth ***and*** un-bound C1)
4. CCF (C2 accepts connect request)
5. *Control Channel Established*

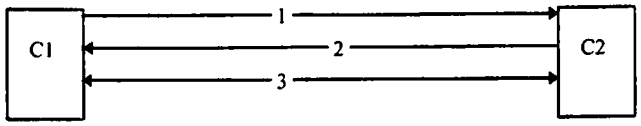
5.3 Caller bound to gatekeeper



(Figure 7)

1. CRQ for C2 (C1 sends connection request for C2)
2. MRQ (G1 Queries C2 for C2's gatekeeper)
3. MRJ (C2 indicates it is not bound to a gatekeeper)
4. CRQ (G1 allows for bandwidth ***and*** un-bound C2)
5. CCF (C2 accepts connect request)
6. CCF
7. *Control Channel Established*

5.4 Neither Client is bound to a gatekeeper



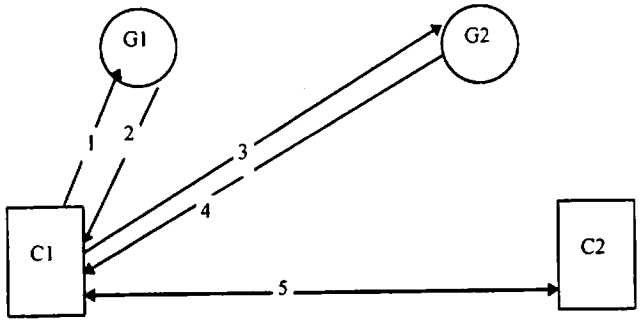
(Figure 8)

1. CRQ for C2 from C1/?? (C1 indicates it has no gatekeeper)
2. CCF (C2 accepts connection without gatekeeper)
3. *Control Channel Established*

6.0 Bandwidth Shifting

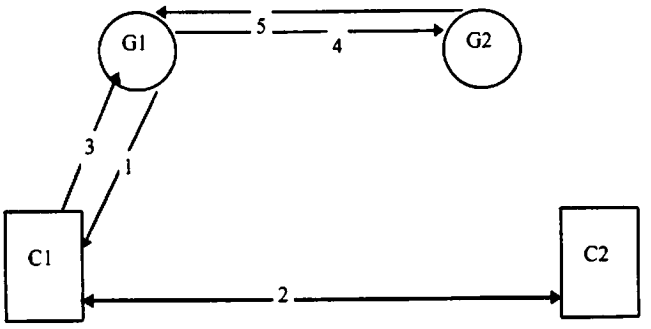
At any time during a conference, the terminals or Gatekeeper may request to increase or decrease the respective bandwidth. The H.245 capabilities negotiation is outside the realm of this recommendation and may actually occur before, during or after this sequence (with no consequences to the gatekeeper specification).

Shown in figures 9 and 10 are client and Gatekeeper initiated requests. It is expected in most cases that the client initiated, will be 'asking for more' ; the Gatekeeper initiated will be 'asking for less'.



(Figure 9)

1. BRQ
2. BCF / BRJ
3. BRQ
4. BCF / BRJ
5. H.245 terminal CAP negotiation



(Figure 10)

1. BRQ
2. H.245 terminal CAP negotiation
3. BCF (or terminate sequence with BRJ)
4. BRQ
5. BCF / BRJ

A bandwidth reject (**BRJ**) is unlikely in step 5, since this negotiation will most probably be for lower bandwidth. In the eventuality that G1 receives a **BRJ** in step 5, it must notify C1 with a new BRQ restoring bandwidth to its original value.

If the sequence terminates in step 3 with a **BRJ** because one or both terminals couldn't handle a bandwidth reduction, the response by the gatekeeper is implementation specific and outside the realm of the specification. A gatekeeper may choose to terminate the conference by issuing a **DRQ**.

7. PDUs

7.1 Background

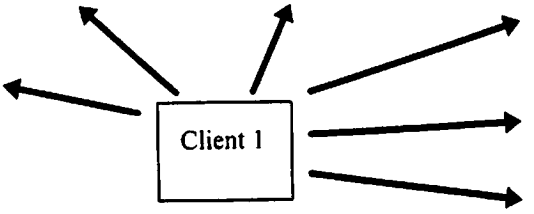
requestSeqNum in PDUs are used to keep track of multiple outstanding requests. It is expected that any associated response PDUs (success or failure) will have the corresponding **requestSeqNum** returned with it.

extensionCount in PDUs is used as a counter to indicate additional bytes following the PDU. For current implementations this value should be initialized to zero (0), to indicate no extension octets present. Future implementations may set this to non-zero to indicate the number of uninterpreted octets that follow.

The **NetworkAddress** structure is meant to capture the various transport formats and includes any transport specific scheme in addition to the possibly local reference to a 'port' number.

```
NetworkAddress              ::=CHOICE
{
        IPAddress           SEQUENCE
        {
                transport   OCTET STRING (SIZE(4)),
                port        INTEGER(0..4294967295)
        },
        IPXAddress          SEQUENCE,
        {
                node        OCTET STRING (SIZE(6)),
                netnum      OCTET STRING (SIZE(4)),
                port        OCTET STRING (SIZE(2))
        },
        IP6Address          SEQUENCE,
        {
                transport   OCTET STRING (SIZE(16)),
                port        INTEGER(0..4294967295)
        },
        NetBios             OCTET STRING (SIZE(16)),
}


        NodeType            ENUMERATED
        {
                Gatekeeper          (1),
                Gateway             (2),
                MC                  (4),
                H323Terminal        (8),
                Undefined Node      (268435456)
        }
```



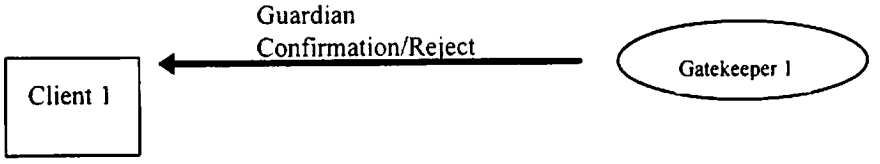
GuardianRequest ::=SEQUENCE --**(GRQ)**

{

yearOfSpec	OCTET STRING (SIZE(4))
requestSeqNum	INTEGER (1..65535),
terminalIdentifier	OCTET STRING (SIZE(128)),
terminalAddress	NetworkAddress,
terminalType	NodeType,
gatekeeperIdentifier	OCTET STRING (SIZE(64)),
extensionCount	INTEGER (0..65535)

}

requestSeqNum - this is a monotonically increasing number unique to the caller. It should be returned by the called in any PDUs associated with this specific PDU.
terminalIdentifier - this is a terminal/user specific string used to identify the caller. It is presumed that application software has made appropriate authentication and this can be 'trusted'. It should be passed unmolested from application end to end.
ControlAddress - this is the network control address for this terminal. If multiple transports are supported, they must be requested separately. This address includes local port information.
terminalType - this specifies the type(s) of the terminal that is registering (note that a H.323 terminal may also be an H.323 MC).
gatekeeperIdentifier - this is a string value that is used to logically identify a called gatekeeper. It should be initialize to a zero (0) value by the caller.



GuardianConfirmation ::=SEQUENCE --**(GCF)**

{

yearOfSpec	OCTET STRING (SIZE(4))
requestSeqNum	INTEGER (1..65535),
gatekeeperIdentifier	OCTET STRING (SIZE(64)),
gatekeeperAddress	NetworkAddress,
extensionCount	INTEGER (0..65535)

}

requestSeqNum - This should be the same value that was passed in the GRQ by the caller.
gatekeeperIdentifier - this is a string value that is used to logically identify a called gatekeeper. It may be used by the caller for future RRQs.
gatekeeperAddress - this is an array of transport addresses; one for each transport that the gatekeeper will respond to. This address includes local port information.

Guardian Reject ::=SEQUENCE **--(GRJ)**

{

yearOfSpec	OCTET STRING (SIZE(4))
requestSeqNum	INTEGER (1..65535),
gatekeeperIdentifier	OCTET STRING (SIZE(64)),
rejectReason	GuardianRejectReason,
extensionCount	INTEGER (0..65535)

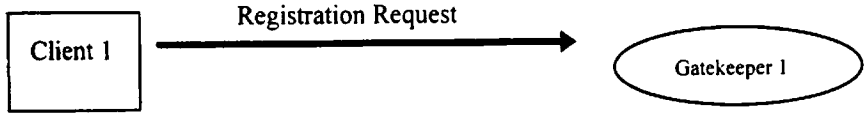
}

GuardianRejectReason ENUMERATED

{

Resource Unavailable	(1),
Terminal Excluded	(2),
Invalid Revision	(5),
Undefined Reason	(65535)

}



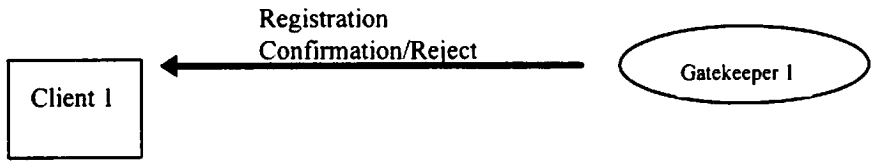
RegistrationRequest ::=SEQUENCE **--(RRQ)**

{

yearOfSpec	OCTET STRING (SIZE(4))
requestSeqNum	INTEGER (1..65535),
bindRequest	BOOLEAN,
terminalIdentifier	OCTET STRING (SIZE(128)),
ControlAddress	NetworkAddress,
terminalType	NodeType,
terminalExtNum	E.164Address
extensionCount	INTEGER (0..65535)

}

requestSeqNum - this is a monotonically increasing number unique to the caller. It should be returned by the called in any PDUs associated with this specific PDU.
bindRequest - set to TRUE if requesting a new binding with called gatekeeper; set to FALSE if registering only.
terminalIdentifier - this is a terminal/user specific string used to identify the caller. It is presumed that application software has made appropriate authentication and this can be 'trusted'. It should be passed unmolested from application end to end.
ControlAddress - this is the network control address for this terminal. If multiple transports are supported, they must be registered separately. This address includes local port information.
terminalType - this specifies the type(s) of the terminal that is registering (note that a H.323 terminal may also be an H.323 MC).
terminalExtNum - This optional value is a phone number by which external (to the LAN) terminals may identify this terminal.



RegistrationConfirmation ::=SEQUENCE --**(RCF)**

{

yearOfSpec	OCTET STRING (SIZE(4))
requestSeqNum	INTEGER (1..65535),
gatekeeperIdentifier	OCTET STRING (SIZE(64)),
gatekeeperAddress	NetworkAddress (SIZE(3)),
extensionCount	INTEGER (0..65535)

}

requestSeqNum - This should be the same value that was passed in the RRQ by the caller.
gatekeeperIdentifier - this is a string value that is used to logically identify a called gatekeeper. It may be used by the caller for future RRQs.
gatekeeperAddress - this is an array of transport addresses; one for each transport that the gatekeeper will respond to. This address includes local port information.

Registration Reject ::=SEQUENCE --**(RRJ)**

{

yearOfSpec	OCTET STRING (SIZE(4))
requestSeqNum	INTEGER (1..65535),
rejectReason	RejectReason,

}

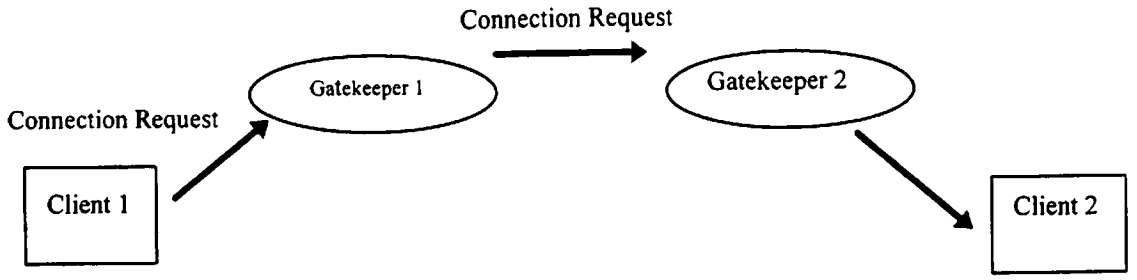
requestSeqNum - This should be the same value that was passed in the RRQ by the caller.

RejectReason ENUMERATED

{

Not Bound Registration	(1),
Duplicate Registration Request	(2),
Invalid Ext Num	(3),
Duplicate Bind Request	(4),
Invalid Revision	(5),
Invalid Network Address	(6),
Undefined Reason	(65535)

}



Connection Request ::=SEQUENCE --**(CRQ)**

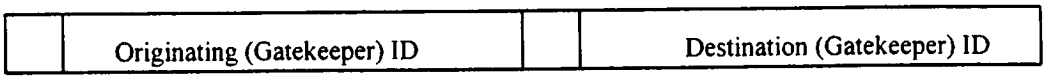
{

yearOfSpec	OCTET STRING (SIZE(4)),
requestSeqNum	INTEGER (1..65535),
originatingID	OCTET STRING (SIZE(128)),
originatingAddress	NetworkAddress,
destinationAddress	NetworkAddress,
originatingGatekeeper	NetworkAddress,
destinationGatekeeper	NetworkAddress,
destinationWanInfo	WanInfo,
conferenceID	INTEGER(0..4294967295),
callType	CallType,
callMedia	CallMedia,
bandWidth	INTEGER (1..4294967295),
connectionID	INTEGER (1..4294967295),
extensionCount	INTEGER (0..65535)

}

requestSeqNum - this is a monotonically increasing number unique to the caller. It should be returned by the called in any PDUs associated with this specific PDU.
originatingID - this is a terminal/user specific string used to identify the caller. It is presumed that application software has made appropriate authentication and this can be 'trusted'. It should be passed unmolested from application end to end. May be used for 'caller-id' functionality.
originatingAddress - this is a specific transport address on which the caller would like a response to this request. It is assumed that this will be the incoming port address for logical channel 0, if this connection is successful.
destinationAddress - this is a specific transport address on which the caller would like to contact the called terminal.
originatingGatekeeper - this is a gatekeeper address to which the caller is bound. If the caller is not bound to a gatekeeper, this should be initialized to all zeros (0).
destinationGatekeeper - this is a gatekeeper address to which the called may be bound. It should be set to zeros (0) by the caller. It may be filled in by a gatekeeper, or the called.
destinationWanInfo - this specifies further contact information that a gateway might use.
conferenceID - this is used to indicate whether this is the initial connection of a new conference or the connection to an existing conference. The value will be set to zero (0) if caller does not wish to partake in a pre-existing conference. If it is non-zero, it specifies the conference which is being joined/invited to. (in which case the connection is to/from a node with MC capabilities)
callType - Using this value, gatekeeper can make determine 'real' bandwidth usage.
callMedia - Utilized by gatekeepers and called to determine acceptance of connection.
bandWidth - the number of 1k BITS/sec requested for the connection.

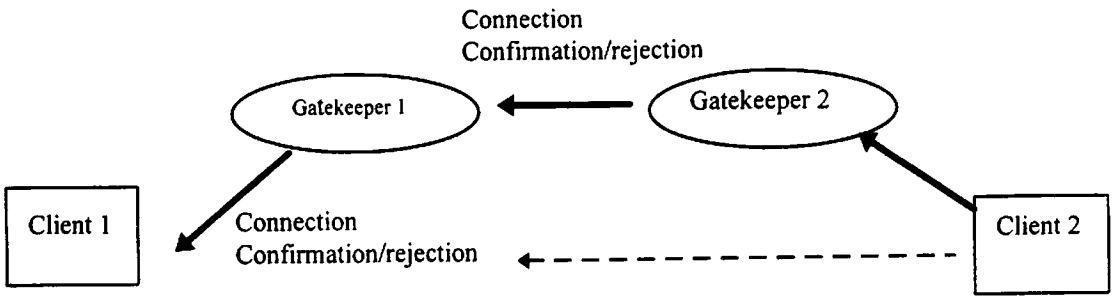
connectionID - Will contain a unique number pair, as specified by the coordinating gatekeepers. The originating and destination gatekeepers will assign values (15 bits) in the LSW and MSW respectively. This will allow the connection to be uniquely identified from all others. In the event that one end or the other does not have a gatekeeper, the msb (bit 16) of the respective LSW/MSW will be set and a terminal supplied number will be assigned.




```
WanInfo                 ::=SEQUENCE
{
        E.164Numbers    SET SIZE(0..6) OF OCTET STRING (SIZE(16)),
        channelRate     INTEGER (1..4294967295),
        ????MORE INFO
}

CallType                        ENUMERATED
{
        PointToPoint            (1)             -- Point to point
        OneToN                  (2),            -- no interaction (a podium)
        NToOne                  (4),            -- no interaction (a listener)
        NToN                    (8),            -- interactive
        BroadCast               (16),           -- Multicast included
}

CallMedia                       ENUMERATED
{
        Data            (1),            -- note that these may be logically OR'd
        Audio           (2),
        Video           (4)
}
```



Connection Confirmation ::=SEQUENCE --**(CCF)**

{

yearOfSpec	OCTET STRING (SIZE(4))
requestSeqNum	INTEGER (1..65535),
originatingGatekeeper	NetworkAddress,
destinationAddress	NetworkAddress,
destinationGatekeeper	NetworkAddress,
conferenceID	INTEGER(0..4294967295),
connectionID	INTEGER (1.. 4294967295),
extensionCount	INTEGER (0..65535)

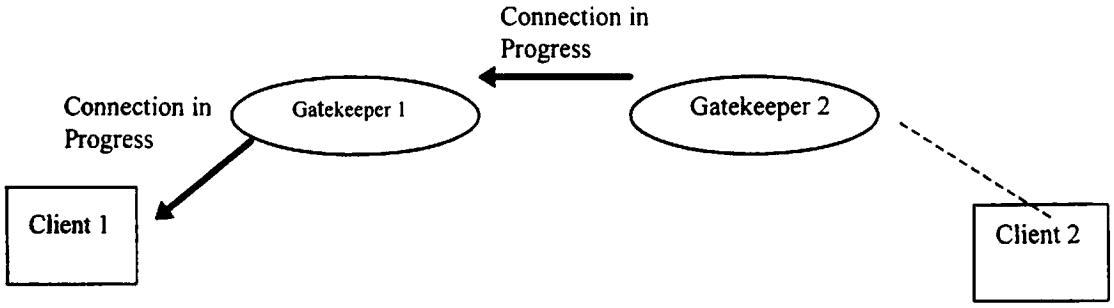
}

requestSeqNum - This should be the same value that was passed in the CRQ by the caller.
originatingGatekeeper - this is a gatekeeper address to which the caller is bound. If the caller is not bound to a gatekeeper, this should be initialized to all zeros (0).
destinationAddress - this is a specific transport address on which the called would like to establish the connection. This may or may not be the same value that was passed in the CRQ.
destinationGatekeeper - If the called is bound to a gatekeeper, this value will contain a valid address.
conferenceID - this is used to indicate whether this is the initial connection of a new conference or the connection to an existing conference. If the value is set to zero (0) indicating that the caller does not wish to partake in a pre-existing conference, and the callee returns an ID associated with the newly created conference.
connectionID - Will contain a unique number pair, as specified by the coordinating gatekeepers. The originating and destination gatekeepers will assign values (15 bits) in the LSW and MSW respectively. This will allow the connection to be uniquely identified from all others. In the event that one end or the other does not have a gatekeeper the bit 16 (msb) of the respective LSW/MSW will be set. The setting of bit 32 will indicate that the MSWhas been assigned a possibly, non-unique value by the called terminal.

```
Connection Rejection        ::=SEQUENCE --(CRJ)
{
        yearOfSpec          OCTET STRING (SIZE(4))
        requestSeqNum       INTEGER (0..65535),
        rejectReason        RejectReason,
        conferenceID        INTEGER(0..4294967295),
        connectionID        INTEGER (1.. 4294967295),
        extProtocolType     ProtocolType,
        extReason           INTEGER(0..65535)
        bandWidth           INTEGER (1..4294967295)  -- measured in 1k bit increments
        extensionCount      INTEGER (0..65535)
}
```

requestSeqNum - This should be the same value that was passed in the CRQ by the caller.
rejectReason - contains numerical reason code for failure of CRQ.
extProtocolType - this is used to specify the protocol family that supplied the reason codes in **extReason**. (currently only Q.931)
extReason - this is the reason code as stipulated by **extProtocolType**.
bandWidth - if **rejectReason** indicates no bandwidth, then this value will contain the maximum that can be requested. This does not protect against a race condition; the caller will have to reissue the CRQ with this lower value, which may again fail. If **rejectReason** is something other than bandwidth, this should be set to 0 and passed un-interpreted.

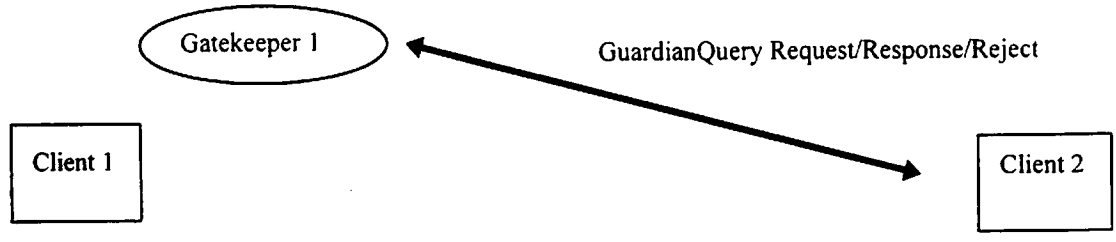
```
RejectReason            ENUMERATED
{
        No Bandwidth            (1),
        Gatekeeper Resources    (2),
        Unreachable Destination (3),
        Destination Rejection   (4),
        Invalid Revision        (5),
        No Permission           (6),
        UnreachableGatekeeper   (7),
        Destination Busy        (8),
        Not Bound               (9),     -- From local Gatekeeper
        Gateway Resources       (10),
        Bad Format Address      (11),
        Caller Not Bound        (12),    -- Destination Gatekeeper
        Caller Not Bound        (13),    -- Destination Gatekeeper
        Destination NoAnswer    (14),
        Undefined Reason        (65535)
}
ProtocolType            ENUMERATED
{
        Q.931                   (1),
        Undefined Protocol      (65535)
}
```



Connection In Progress ::=SEQUENCE **--(CIP)**

```
{
        yearOfSpec              OCTET STRING (SIZE(4))
        requestSeqNum           INTEGER (1..65535),
        connectionID            INTEGER (1.. 4294967295),
        channelNum              INTEGER (0..65535),          -- ID of WAN channel #
        connection Status       ConnectionStatus
        extensionCount          INTEGER (0..65535)
}

        ConnectionStatus        ENUMERATED
        {
                Connected               (0),
                No Connection           (1),
                Idle                    (2),
                Disconnecting           (3),
                Dialing                 (4),
                Connecting              (5),
                Ringing                 (6),
                Redirecting             (7),
                Undefined Status        (65535)
        }
```



Guardian Query Request ::=SEQUENCE **--(GQQ)**

```
{
        yearOfSpec              OCTET STRING (SIZE(4))
        requestSeqNum           INTEGER (1..65535),
        replyAddress            NetworkAddress,
        extensionCount          INTEGER (0..65535)
}
```

Guardian Query Response ::=SEQUENCE --**(GQRS)**

{

yearOfSpec	OCTET STRING (SIZE(4))
requestSeqNum	INTEGER (1..65535),
gatekeeperIdentifier	OCTET STRING (SIZE(64)),
gatekeeperAddress	NetworkAddress,
extensionCount	INTEGER (0..65535)

}

Guardian Query Reject ::=SEQUENCE --**(GQRJ)**

{

yearOfSpec	OCTET STRING (SIZE(4))
requestSeqNum	INTEGER (1..65535),
rejectReason	GuardianQueryRejectReason,

}

GuardianQueryRejectReason ENUMERATED

{

No Gatekeeper	(1),
Invalid Revision	(5),
Undefined Reason	(65535)

}

Status Request ::=SEQUENCE --**(SRQ)**

{

yearOfSpec	OCTET STRING (SIZE(4))
requestSeqNum	INTEGER (1..65535),
connectionID	INTEGER (0..4294967295),
extensionCount	INTEGER (0..65535)

}

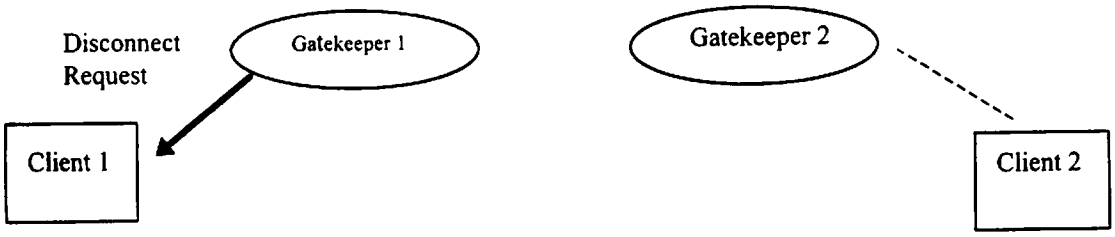
connectionID - This may be set to 0 to indicate the a 'logical' connection is being queried. It is up to local interpretation as to what this means. It may be interpreted as the 'first' connection made, or it may be interpreted as 'all' connections in which case the node may respond with multiple SRRs.

Status Report Response ::=SEQUENCE --**(SRR)**

{

yearOfSpec	OCTET STRING (SIZE(4)),
requestSeqNum	INTEGER (1..65535),
nodeType	NodeType,
conferenceID	INTEGER(0..4294967295),
connectionID	INTEGER (1..4294967295),
callState	ConnectionStatus,
originatingID	OCTET STRING (SIZE(128)),
originatingAddress	NetworkAddress,
destinationAddress	NetworkAddress,
originatingGatekeeper	NetworkAddress,
destinationGatekeeper	NetworkAddress,
destinationWanInfo	WanInfo,
callType	CallType,
callMedia	CallMedia,
bandWidth	INTEGER (1..4294967295),
bytesSent	INTEGER (1..4294967295),
bytesRcvd	INTEGER (1..4294967295),
extensionCount	INTEGER (0..65535)

}



DisconnectRequest ::=SEQUENCE --**(DRQ)**

{

yearOfSpec	OCTET STRING (SIZE(4))
requestSeqNum	INTEGER (1..65535),
connectionID	INTEGER (1.. 4294967295),
terminalIdentifier	OCTET STRING (SIZE(128)),
gatekeeperIdentifier	OCTET STRING (SIZE(64)),
disconnectReason	DisconnectReason,
extProtocolType	ProtocolType,
extReason	INTEGER(0..65535)
reasonString	OCTET STRING (SIZE(80)),
delayTime	INTEGER (0..65535), - number of seconds
extensionCount	INTEGER (0..65535)

}

This PDU may be sent between peer terminals or Gatekeepers, and under usual circumstances will be issued by a terminal to its guardian Gatekeeper. It is assumed that *if* this PDU is sent from a Gatekeeper to a terminal that the terminal will in turn, send a like PDU to its bound Gatekeeper. This will allow the tear-down of associated, local logical channels.

DisconnectReason ENUMERATED

{

Hang Up	(1),
Remote Hang Up	(2),
Remote Abort	(3),
Transfer	(4),
Gatekeeper	(5)
Undefined Reason	(65535)

}

connectionID - ID of connection that is to be disconnected.
extProtocolType - this is used to specify the protocol family that supplied the reason codes in **extReason**. (currently only Q.931)
extReason - this is the reason code as stipulated by **extProtocolType**.
reasonString - a string value that is optionally supplied by the caller as to why the disconnection has been requested. If this PDU is from the Gatekeeper to a terminal, this can be 'displayed'.
delayTime - a count in seconds before the actual disconnect will be initiated. This can be used by a Gatekeeper when 'forcing' a terminal to disconnect.

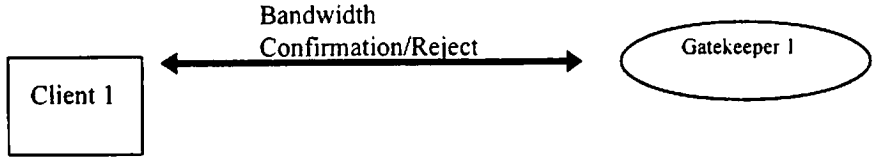
BandwidthRequest ::=SEQUENCE **--(BRQ)**

{

yearOfSpec	OCTET STRING (SIZE(4))
requestSeqNum	INTEGER (1..65535),
terminalIdentifier	OCTET STRING (SIZE(128)),
connectionID	INTEGER (1.. 4294967295),
callType	CallType,
callMedia	CallMedia,
replyAddress	NetworkAddress,
gatekeeperIdentifier	OCTET STRING (SIZE(64)),
bandWidth	INTEGER (1..4294967295) -- measured in 1k bit increments
extensionCount	INTEGER (0..65535)

}

requestSeqNum - this is a monotonically increasing number unique to the caller. It should be returned by the called in any PDUs associated with this specific PDU.
terminalIdentifier - this is a terminal/user specific string used to identify the caller/called. It is presumed that application software has made appropriate authentication and this can be 'trusted'.
connectionID - ID of connnection that is to have the bandwidth changed.
callType - Using this value, gatekeeper can make determine 'real' bandwidth usage.
callMedia - Can be utilized by gatekeepers and called to determine acceptance of connection.
replyAddress - this is the transport address to which the **BCF**, or **BRJ** is to be sent.
gatekeeperIdentifier - this is a string value that is used to logically identify a calling/called gatekeeper.
bandWidth - the NEW number of 1k BITS/sec requested for the connection.



Bandwidth Confirmation ::=SEQUENCE **--(BCF)**

{

yearOfSpec	OCTET STRING (SIZE(4))
requestSeqNum	INTEGER (1..65535),
bandWidth	INTEGER (1..4294967295) -- measured in 1k bit increments
extensionCount	INTEGER (0..65535)

}

requestSeqNum - This should be the same value that was passed in the **BRQ** by the caller.
bandWidth - the maximum that *might* be offered with a new **BRQ**.

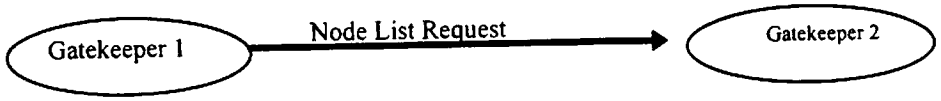
```
Bandwidth Reject                ::=SEQUENCE --(BRJ)
{
        yearOfSpec              OCTET STRING (SIZE(4))
        requestSeqNum           INTEGER (1..65535),
        rejectReason            BandRejectReason,
}

        BandRejectReason        ENUMERATED
        {
                Not Bound               (1),
                Invalid ConnectionID    (2),
                Invalid Permission      (3),
                Request Denied          (4),
                Invalid Revision        (5),
                Undefined Reason        (65535)
        }
```

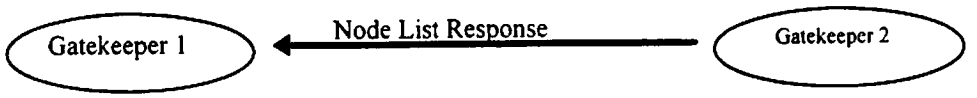
Nodes are typed into four broad categories: Gatekeepers, Gateways, MCs, and terminals. Different types may be returned within the list supplied in the NRL. In all cases, the responding node (that issuing the NRL; currently only Gatekeepers) will include their address information. It should be noted that all information in **NodeEntry** may not be relevant depending on the terminal type.

A protocol for the discovery and tracking of gatekeeper/gateway/mc/terminal nodes is beyond the scope of this specification. One possibility is that gatekeepers use a 'hello' mechanism (perhaps a NLR containing only themselves) and cache this along with any terminal registrations. For example, gatekeepers may age out cached remote node information if a configurable number of periods have passed without receiving a NRL.

1. Node List Request (NLR). Any gatekeeper can issue this broadcast (or multicast) at any time. All gatekeepers receiving the request will respond with their gatekeeper address information and a list of all other known nodes (NRL).
2. Node Response List (NRL). This can be sent point to point in response to a received NLR. It may also be periodically broadcast (or multicast).




```
NodeListRequest                 ::=SEQUENCE --(NLR)
{
        yearOfSpec              OCTET STRING (SIZE(4))
        requestSeqNum           INTEGER (1..65535),
        gatekeeperIdentifier    OCTET STRING (SIZE(64)),
        gatekeeperAddress       NetworkAddress,
        nodeType                NodeType
        extensionCount          INTEGER (0..65535)
}
```




```
NodeResponseList            ::=SEQUENCE --(NRL)
{
        yearOfSpec          OCTET STRING (SIZE(4))
        requestSeqNum       INTEGER (1..65535),
        responseStatus      ResponseStatus
        gatekeeperIdentifier OCTET STRING (SIZE(64)),
        nodeList            SEQUENCE SIZE(1..256) OF NodeEntry,
        extensionCount      INTEGER (0..65535)
}
        ResponseStatus      ENUMERATED
        {
                Success                 (0),
                Not Supported           (1),
                Unknown Node Type       (2),
                Invalid Revision        (5),
                Undefined Status        (65535)
        }

        NodeEntry           ::=SEQUENCE
        {
                nodeIdentifier      OCTET STRING (SIZE(64)),
                nodeSeqNum          INTEGER (1..65535)),
                nodeAge             INTEGER (1.. 4294967295),
                nodeAddress         NetworkAddress (SIZE(3)),  -- one per transport
                nodeType            NodeType
        }
```

8. Other Considerations

8.1 Gatekeeper to Gatekeeper Information Exchange

The gatekeepers will maintain information about other gatekeepers in the enterprise. This is done primarily for presenting a list of gatekeepers to the admin to allow for remote control and management.

We are concerned that we not reinvent the wheel. Gatekeepers should be manageable in the enterprise via SNMP. *There appears to be no standardized method for node discovery in SNMP. SNMP manageable entities are discovered via "Hello" broadcasts and ARPs (TCP/IP specific solution).* Further investigation required here.

Gatekeeper address information is exchanged. Optionally gateway/MC information may also be exchanged. The capability to keep gateway information private should be provided. Note that client information is not included. Client information will be exchanged during normal operation (described in section 4). There are two PDUs for accomplishing this information exchange:

1. Node List Request (NLR). Any gatekeeper can issue this broadcast (or multicast) at any time. The NLR will specify whether gatekeeper or gateway information is requested. In the event of a gatekeeper list request, all gatekeepers receiving the request will respond with their gatekeeper address information and a list of all other known gatekeepers (NRL). In the event of a gateway list request, the gatekeepers receiving the request may respond with their gateway address information and a list of all other known gateways.
2. Node Response List (NRL). This can be sent point to point in response to a received NLR. It is also periodically broadcast (or multicast). The periodical interval shall be configurable (default = 3 hours).

Gatekeepers will age out cached remote gatekeeper information if a configurable number of periods (default = 4) have passed without receiving a NRL (default = 12 hours).

When a gatekeeper is first activated, it sends an immediate NRL broadcast (or multicast) which contains just its own address. This is used as a notification that a new gatekeeper has joined the enterprise. This is also used for identifying rogue conferences (see below). It then sends a NLR and caches the information returned in the point to point NRLs with the other gatekeeper and gateway information.

NOTE: The gatekeeper broadcast is on a different port than the client broadcast (see section 3.1). It is strongly recommended that IT organizations configure their routers to forward this broadcast (versus blocking the client broadcast).

8.2 Rogue conference discovery

If both terminals are bound to gatekeepers, they send unsolicited status (SRR) every 5 minutes to their gatekeepers for the duration of the conference. If one or both terminals are not bound to a gatekeeper they cannot send SRRs.

When a new gatekeeper is started, it sends a broadcast (or multicast) announcing its presence. Unmanaged terminals that are in a conference will detect this broadcast (or multicast) and start sending unsolicited status messages to the new gatekeeper. These SRR messages will identify the conference as a "Rogue". It is up to the gatekeeper what action to take upon discovering rogue conferences.

> A gatekeeper can send a status request (SRQ) at any time to a client. The client will respond with an immediate SRR. This allows an updated snapshot of conference activity.