Department of Computing Science and Mathematics
University of Stirling

Policies for H.323 Internet Telephony

Tingxue (Sean) Huang

Technical Report CSM-165

ISSN 1460-9673

May 04

Department of Computing Science and Mathematics
University of Stirling

Policies for H.323 Internet Telephony

Tingxue (Sean) Huang

Department of Computing Science and Mathematics
University of Stirling
Stirling FK9 4LA, Scotland
Telephone +44-1786-467421, Facsimile +44-1786-464-551

Email th@cs.stir.ac.uk

Technical Report CSM-165

ISSN 1460-9673

May 04

Abstract

Firstly, this report examines in which mode an H.323 gatekeeper should work for enforcing policies. Then, it explores how the gatekeeper can cooperate with a policy server. The report discusses how to complement a gatekeeper with policies, using GNU Gk as the basis. The approach takes into account issues of robustness, simplicity and scalability. Finally, the report investigates how to design H.323 policies, and how to combine these with SIP policies on a policy server.

Keywords : Feature, Gatekeeper, GNU Gk, H.323, Internet Telephony, Policy

Table of Contents

List of Figures

List of Tables

1 Introduction

This report talks about developing policy support for H.323 communications. At the moment, the separate project named ACCENT (Advanced Call Control Enhancing Network Technologies) is developing policy-based control of Internet Telephony. It is creating protocol-independent support of communication policies, and an instantiation in a SIP environment. This report exploits the same policy infrastructure but for an H.323 network.

1.1 Motivation

As communication technologies and computer science develop, the Internet is affecting people's lives in many fields. IP telephony is one of the most important applications. It goes against traditional telecomms on the grounds of convenience and cheapness. An IP telephony system provides many kinds of features such as verbal conversation, video calls, voice email, fax and *ad hoc* conferencing. It is very convenient because users can choose any communication device to receive any form of messages. So, some people refer to IP telephony as a UMS (Unified Messaging System), and refer to its services as three 'A's of communications: any place, any device and any time.

However, IP telephony also causes a problem. Because it is almost too convenient, it often intrudes upon users. When any call or message arrives, the user is informed. As a result, the user becomes the slave of the IP telephony system. Therefore, it is necessary that users be able to control their availability. In the traditional telecomms system, users have few abilities to control this: users are passive when it comes to communication. In the IP telephony network, users can control their availability on the basis of where they are, who the caller is, the type of the call, the time of day, and so on.

So, the goal of this work has been to introduce policies into IP telephony. Some papers [3, 4] have discussed policies applied to a SIP (Session Initiation Protocol) system. A special language has been defined for describing policies [10]. In the IP telephony field, H.323 is another popular multimedia communications protocol. Many IP telephony networks operate using this. So, it makes sense to explore policies for H.323 networks as well.

1.2 Scope and Objectives

H.323 is not a single protocol but a protocol group. It defines all kinds of communications modes for many situations. In order to implement H.323 policies, an H.323 network must work in the appropriate mode. A gatekeeper is the management device of an H.323 zone, and so is the best candidate for enforcing H.323 policies. The gatekeeper is required to obtain most of the call information for H.323 policies. So, the mode in which the gatekeeper should work is very relevant to enforcing H.323 policies.

A policy exists at a higher level than a feature. The H.450 recommendation specifies more than ten supplementary services, that is to say, features. Our telecommunications system implements these features through policies. The H.323 supplementary services are therefore reformulated as policies. To support H.323 policies, the cooperation between an H.323 gatekeeper and a policy server must be explored. The gatekeeper must abide by the H.323 protocol when enforcing H.323 policies. A policy module has been developed for the GNU Gk gatekeeper. The development of this addressed issues of robustness, simplicity and scalability.

The research has had two aspects: enhancement of the gatekeeper, and enhancement of the policy server. An H.323 network shares a common policy server with a SIP network. In order to get greater efficiency, the policy server is stand-alone and independent of the underlying communications network. Regarding the policy server, the ACCENT project has defined an architecture that deals with storing policies, retrieving policies, filtering policies, and solving policy interactions. ACCENT is still developing the interface with SIP network. For H.323 policies, the policy server must have a special interface to an H.323 network and H.323-oriented policies.

1.3 Overview of Report

This report focuses on several aspects of H.323 policies. Some related documents discusses the protocol-independent policy server [3, 4] and the definition of the policy language [12]. The reader should consider these as supporting documentation. In addition, ACCENT-related information is available from www.cs.stir.ac.uk/compass/. The report discusses four aspects of H.323 policies.

Section 2 explores in which mode the gatekeeper should work for enforcing H.323 policies. This section lists the four working mode of gatekeeper. In order to implement H.323 policies, the gatekeeper needs to monitor most of the call information. Therefore, we require the gatekeeper to operate in proxy mode. The whole H.323 call is set up in three stages: call admission (RAS), call signalling (Q.931) and call control (H.245). All the call messages pass through the gatekeeper. In addition, the RTP media channels and T.120 data channels also go through the gatekeeper. Not only does the gatekeeper request and receive policy actions, but also it can enforce some special policies through operating the media channels and data channels.

Section 3 discusses ten policy trigger events such as registration, no answer, bandwidth request and so on. Then, based on these trigger events, more than ten H.323 policies are explored including a registration policy, a forward-always policy, a bandwidth policy and so on. The report explores what information extracted from the call message for each kind of policy, which possible result policy responses and the message flow of each kind of policy.

Section 4 uses an open source gatekeeper called Gnu Gk to implement H.323 policies. Firstly, we dissect GNU Gk through exploring its class inheritance, ER diagram and operations. Then two classes, *PolicyCallSignalSocket* and *PolicyH323RasSrv*, are defined to inherit from *CallSignalSocket* and *H323RasSrv* respectively. They capture policy trigger events, extract call information, pass this to a policy server, receive and implement the policy response.

Section 5 discusses the H.323 module for the policy server. Firstly, the interface between the policy server and the gatekeeper is explored. The differences from the SIP policy module are explored. Then, the terminology mapping table is discussed that maps between the H.323 protocol and policy concepts. In addition, it is explained how to define H.323 policies. Finally, the H.323 processing flow is presented.

1.4 H.323 Terminology

H.323 terms are explained briefly below.

RAS (Registration, Admission and Status)

RAS refers to H.225.0, including lots of messages such as *RRQ/RCF/RRJ* and *BRQ/BCF/BRJ*. These are transmitted through the RAS signalling channel during the call. They are used to perform registration, admission, bandwidth change, status, and disengage procedures between endpoints and gatekeepers.

Gatekeeper Request (GRQ), Gatekeeper Confirm (GCF), Gatekeeper Reject (GRJ)

These messages are used to discover the gatekeeper automatically. An endpoint (including a terminal, an Multipoint Control Unit and a gateway) looks for its own gatekeeper by sending a *GRQ* message. The gatekeeper responds to the endpoint with a *GCF* message (confirm) or a *GRJ* message (reject).

A *GRQ* message contains some important element fields such as the type of the endpoint, the transport address for the endpoint, and the identifier of the gatekeeper. The *GCF* message includes the identifier of the gatekeeper, the transport address for the gatekeeper, a sequence of prioritised alternatives for the gatekeeper, etc. A *GRJ* message has two necessary items: the identifier of the gatekeeper and a reject reason.

Registration Request (RRQ), Registration Confirm (RCF), Registration Reject (RRJ)

These messages are used for registration by a terminal or a gateway.

An *RRQ* message is a request from a terminal to a gatekeeper to register. It contains some important items such as the call signalling transport address for the endpoint, the registration and status transport address for the endpoint, the type of the endpoint, a list of alias addresses for the endpoint, the identifier of the gatekeeper, the endpoint, and the endpoint's maximum and current call capacity.

If the gatekeeper responds with an *RCF*, the terminal will use the corresponding gatekeeper for future calls. It contains the following fields: an array of transport addresses for H.225 call signalling messages, a list of alias addresses for this terminal, the gatekeeper's identifier, the endpoint's identifier, and a list of prefixes by which other endpoints may identify this endpoint.

If the gatekeeper responds with an *RRJ*, the terminal must seek another gatekeeper to register with. The *RRJ* includes two important items: the reason for the rejection of registration, and the gatekeeper's identifier.

Admission Request (ARQ), Admission Confirm (ACF), Admission Reject (ARJ)

These messages are used to request access to the packet-based network. An *ARQ* message requests access. The gatekeeper grants the request with an *ACF* or denies it with an *ARJ*.

An *ARQ* message includes the following important fields: the endpoint's identifier, a sequence of alias addresses for the destination, the transport address used at the destination for call signalling, the external address for multiple calls, a sequence of alias addresses for the source endpoint, the transport address used at the source for call signalling, the bidirectional bandwidth requested for the call, and a globally unique call identifier set by the originating endpoint.

An *ACF* message contains the allowed maximum bandwidth for the call, the transport address to which to send Q.931 call signalling, the address for the initial channel, and possible additional channel calls.

An *ARJ* message includes one necessary item: the reason the admission request is denied.

Bandwidth Request (BRQ), Bandwidth Confirm (BCF), Bandwidth Reject (BRJ)

These messages are used to change the packet-based network bandwidth. Both the endpoint and the gatekeeper can ask to raise or lower the bandwidth with a *BRQ* message. The *BCF* message is used to grant the request, or the *BRJ* message is used to deny the request.

A *BRQ* message contains the endpoint's identifier, the new bidirectional bandwidth requested for the call, a globally unique call identifier, and the gatekeeper's identifier. A *BCF* message includes one important item named *bandWidth* that specifies the maximum allowed at this time. A *BRJ* message contains one necessary element, *rejectReason*, specifying the reason the change was rejected.

Location Request (LRQ), Location Confirm (LCF), Location Rejection (LRJ)

These messages are used to ask a gatekeeper to provide address translation. An endpoint can send an *LRQ* to a gatekeeper, and a gatekeeper can also send this to another gatekeeper. The requested gatekeeper responds with an *LCF* containing the transport address of the destination, or rejects the request with *LRJ*.

Disengage Request (DRQ), Disengage Confirm (DCF), Disengage Rejection (DRJ)

These messages are used to request termination of a call. The DRQ message could be sent by a gatekeeper or an endpoint. A *DRQ* is different from a *ReleaseComplete* message because its purpose is to inform the gatekeeper of the termination of a call. If it is not terminating the call signalling channel, the gatekeeper may not receive the *ReleaseComplete*.

2 H.323 Call Flow for Policies

2.1 H.323 Network Structure

An H.323 communications network is built from many basic units named H.323 zones. An H.323 zone, as Figure 1 (based on [1]), is the collection of all terminals (Tx), Gateways (GW), and Multipoint Control Units (MCUs) managed by a single Gatekeeper (GK). An H.323 zone has one and only one Gatekeeper. An H.323 zone is independent of the network topology, and may comprise multiple network segments that are connected using routers (R) or other devices.

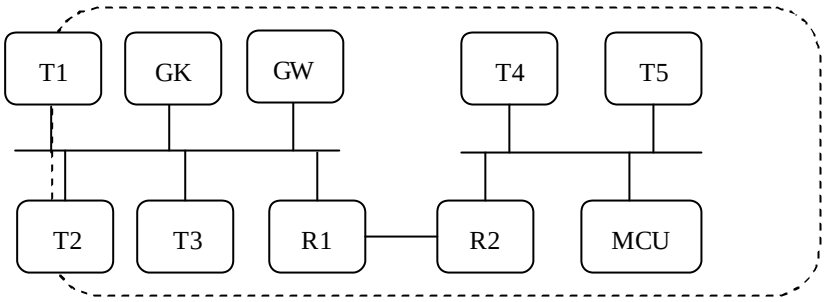

Figure 1. An H.323 Zone

A full H.323 call contains three stages: call admission (RAS), call signalling (Q.931), and call control (H.245). These occur in the given order. Endpoints apart from the gatekeeper, but including terminals, gateways and MCUs, need to send some H.225 RAS messages (e.g. *RRQ*, *ARQ*, *DRQ*) to gatekeepers before or after making a call. Endpoints can also send some RAS messages (e.g. *IRQ/IRR* and *BRQ/BCF*) during a call. Gatekeepers can work in four ways, as shown in Table 1. In mode 1, only RAS messages go through the gatekeeper. In mode 2, both RAS messages and H.225 call signalling messages pass through the gatekeeper. In mode 3, except for the RTP media stream and T.120 data, all other messages go through the gatekeeper; this includes RAS messages, H.225 call signalling messages and H.245 call control messages. In order to cooperate with a policy server, the gatekeeper must extract most of the information about a call.

Mode	H.225 RAS	GK-Routed	H.245-Routed	RTP/RTCP,T.120
1	✓	×	×	×
2	✓	✓	×	×
3	✓	✓	✓	×
Proxy	✓	✓	✓	✓

Table 1. Working Modes of The Gatekeeper

If the gatekeeper just implements the functions that the H.323 protocol defines, it has the first three working modes discussed above. But if the gatekeeper is required to implement some special supplementary services, such as a call intrusion ('barge-in') service, it should have the capability of controlling the media and data stream. In other words, media and data channels also pass through the gatekeeper. The gatekeeper must therefore act as a proxy because it deals with all the H.323 call control messages, media and data channels. In this work it is assumed that the gatekeeper works in proxy mode in order to investigate policy issues thoroughly. In practice, the gatekeeper can be configured according to the specific requirements.

2.2 RAS message flow

Firstly, an endpoint needs to know its gatekeeper. An endpoint can designate its own gatekeeper manually. Alternately, endpoints can use a *GRQ* message to locate a gatekeeper automatically in its own H.323 zone. In the automatic case, the gatekeeper will reply using a *GCF/GRJ* message as shown in Figure 2.

After the endpoint finds its gatekeeper and has registered with the gatekeeper using *RRQ/RCF* messages, the endpoint can make a call. Through the registration procedure, the gatekeeper acquires some information about the endpoint. The information includes some necessary fields such as the address for call signalling, the address for RAS and aliases, and many optional fields such as non-standard data.

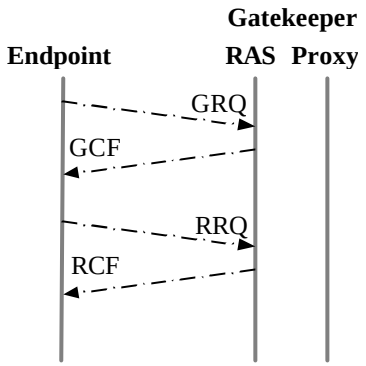

Figure 2. H.225 RAS Message Exchange

After registration an endpoint can make a call through its own gatekeeper.

2.3 H.225 Call Signalling Flow (Q.931)

In order to discuss H.323 policies in Section 3, for the moment we explore the most appropriate gatekeeper mode shown in 0. In fact, the H.323 protocol considers all kinds of cases. For example the caller and the callee might register with the same gatekeeper. The source gatekeeper and the destination gatekeeper might follow any mode listed in Table 1.

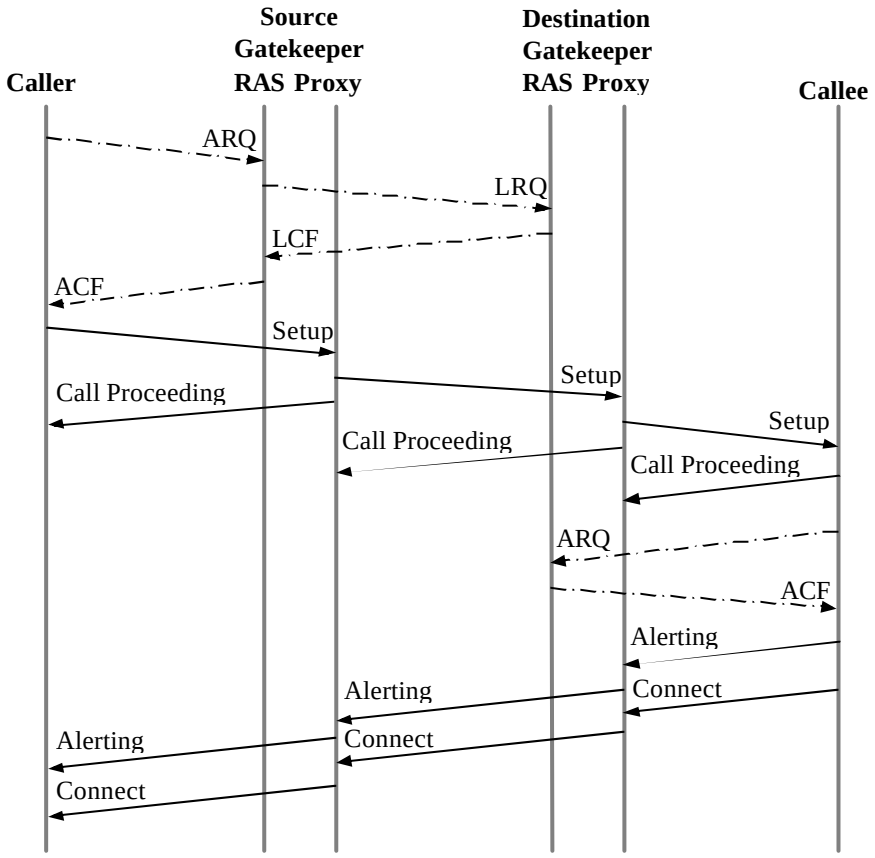

Figure 3. Both Gatekeepers Routing Call Signalling

Firstly, the caller initiates an *ARQ/ACF* exchange with the source gatekeeper. If the gatekeeper cannot resolve the callee's address, it will multicast an *LRQ* to locate the callee. The destination gatekeeper will return an *LCF* with its call signalling transport address. After the caller receives the *ACF* with the source gatekeeper's call signalling transport address, it sends a *Setup* message to the source gatekeeper. Then the source gatekeeper will send a *Setup* message to the destination gatekeeper to be passed on to the callee. The callee initiates an *ARQ/ACF* exchange with the destination gatekeeper. The callee responds to the destination gatekeeper with the *Connect* message that contains its H.245 control transport address for use in H.245 signalling. The destination gatekeeper sends a *Connect* message to the source gatekeeper that contains the destination gatekeeper's H.245

control transport address. The source gatekeeper sends a *Connect* message to the caller that contains its H.245 control transport address.

After this stage, the caller, gatekeepers and the callee know the peer's H.245 address. Then, through the H.245 call control channel, the caller and the callee can exchange their capability set.

2.4 H.245 Control Signalling Flow

After dealing with H.225 call signalling messages, an H.323 call enters the H.245 control stage. The latest H.323 protocol considers the H.245 control stage as an alternative option. Because it takes a long time to initiate an H.323 call, in order to reduce the time for call setup the H.323 protocol provides a *Fast Connect* to replace the H.245 control stage. If a caller tries to use the *Fast Connect*, its *Setup* message will contain the information about the *Fast Connect* and the capability set. The callee can transmit audio and video streams when sending the *Connect* message. Of course, because the *Fast Connect* is a new addition some old H.323 endpoints will not support it. In general, new endpoints will support this capability. To discuss all aspects, H.245 control signalling flow is shown in 0. Because gatekeepers work in the proxy mode listed in Table 1, all H.245 control signalling goes through all gatekeepers.

During the procedures of H.245, the caller and the callee exchange system capabilities by transmitting H.245 *Terminal Capability Set* messages. Then, the endpoints make a master-slave determination. Finally, the endpoints open media channels for audio and/or video using *Open Logical Channel* messages. After this stage, the caller and the callee have RTP/RTCP media channels for transmitting audio and/or video streams.

0 displays the procedure for media transmission and release of an H.323 call. After opening a logical channel, the caller and the callee can transmit the audio and/or video streams using the RTP/RTCP channels. These channels are established through the gatekeeper. The gatekeeper can dispose of the media channels as it deals with the call signalling and control messages. During the call, the T.120 channel for transmitting data can be established. Because the gatekeeper is in proxy mode, the data channel is also through the gatekeeper.

The release procedure is finished by sending H.225 RAS call signalling messages. So, the release signalling messages also go through the gatekeeper. The H.323 protocol specifies that the H.225 call signalling channel can be torn down during the procedure of media transmission or can be kept established. Normally, endpoints will keep the H.225 call signalling channel alive. The caller or the callee will send a *Release Complete* message if it wants to end the call. Finally, both the caller and the callee exchange DRQ/DCF messages with their own gatekeeper.

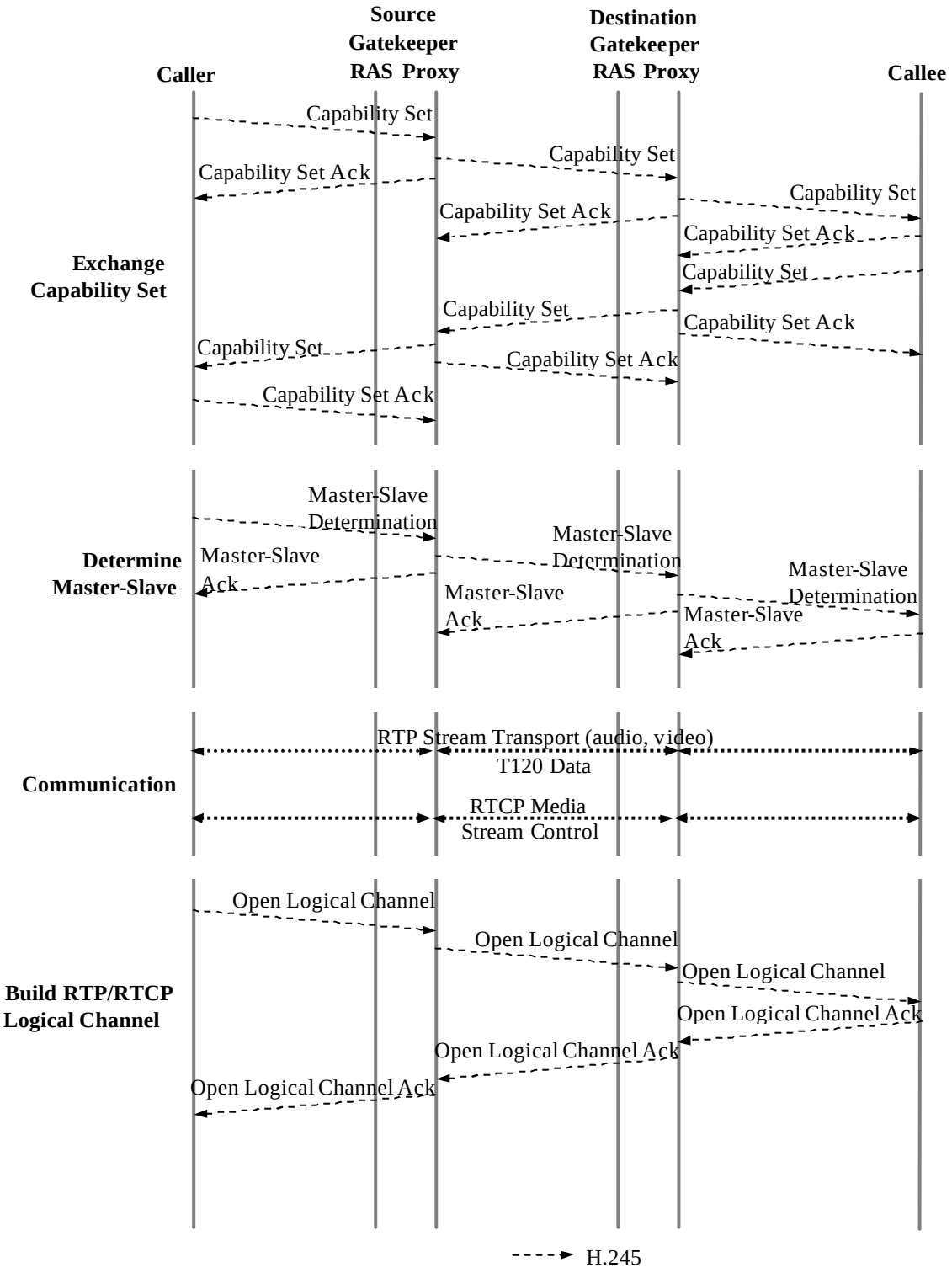

Figure 4. H.245 Control Stage, Communication Stage and Release Complete Stage

3 Cooperation between Gatekeeper and Policy Server

Section 2 described the signalling flow of a basic call. In addition, the H.450 recommendation specifies more than ten supplementary services. These services are mainly invoked in endpoints or gatekeepers using the *Setup* message or the *Facility* message. We can refer to these supplementary services as features. Our communications system implements these features through policies. Some research papers have discussed the important difference between policies and features very clearly [3][4]. Features are described as specific, prescriptive and imperative; they have little scope for individualisation. However, policies are more flexible and generic. They provide a mechanism to express the preferences of subscribers.

Policy servers store and filter policies. A gatekeeper extracts call information and passes it to a policy server. This will look for the corresponding policy in the policy repository, filtering the policies and checking for policy interaction. 0 shows the H.323 policy architecture, and can be compared to the one for SIP [3].

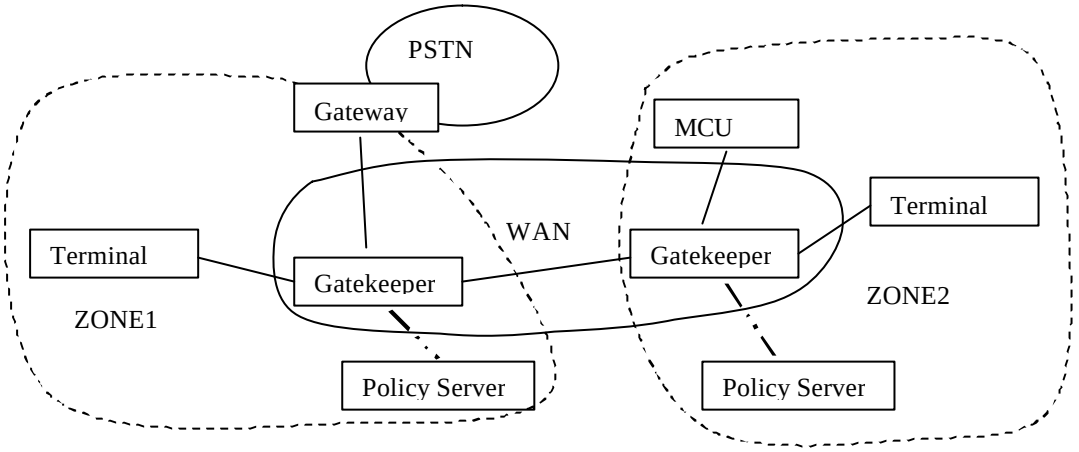

Figure 5. H.323 Policy Architecture

Following the structure of the H.323 call procedures, a variety of policies is presented in the following sections.

3.1 Registration Policy

A registration policy is not truly useful for end users. But some organisations, such as enterprises and universities, require different groups to register different information. For example, university students might have to provide just one alias and signalling transport address for greater efficiency because there are many students. Lecturers, according to their preference, could register with extra information such as more than one alias and *CryptoTokens* for encryption.

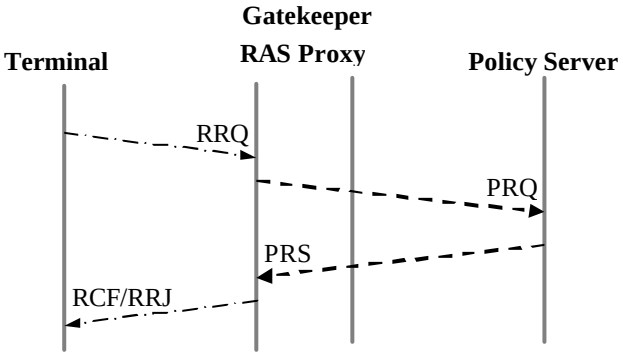

Figure 6. Registration Policy

As shown in Figure 6, after extracting the necessary information from an *RRQ* message, the gatekeeper will pass the information to the policy server in a *PRQ* message. A *PRQ* message contains pairs of the form (*variable*, *value*). From the *RRQ* message, the information extracted by the gatekeeper includes the type of

endpoint, the transport address for RAS, the transport address for call signalling, and one or more aliases. Some optional items include the capability of multiple calls, encryption tokens, and so on. The policy server will reply with a *PRS* message. This indicates whether the gatekeeper should confirm the registration or should reject it for a specified reason.

3.2 Admission Policy

When the caller plans to make a call, it is firstly necessary to send an *ARQ* message to the gatekeeper. When the callee receives the *Setup* message, an *ARQ* message must be sent to the local gatekeeper. Although these messages have the same data structure, they have different functions. The former requests outgoing admission (outgoing-ARQ message) and the latter requests incoming admission (incoming-ARQ message). In order to simplify the presentation shown in Figure 7, we presume that both the caller and the callee register with the same gatekeeper

The gatekeeper extracts some necessary information from an *ARQ* message, including the type of call, the information about destination and source, bandwidth, and the reference value of the call. If some optional items such as alternative destination and source exist, they are also passed to the policy server. The policy server returns a *Continuing* or *Rejecting* action. If it is a *Continuing* action, the gatekeeper will send an *ACF* message to the terminal. If it is a *Rejecting* action, the gatekeeper will send an *ARJ* message with the reason given by the policy server. In fact, different policies may apply to the *outgoing-ARQ* message and the *incoming-ARQ* message. So, the gatekeeper needs to include the user in a *PRQ* message. If the user is the caller, the policy server will return policies for an *outgoing-ARQ* message, and conversely for the callee and an *incoming-ARQ* message. Alternatively, the *PRQ* message can contain a variable named *Trigger* that gives the type of *ARQ* message definitively.

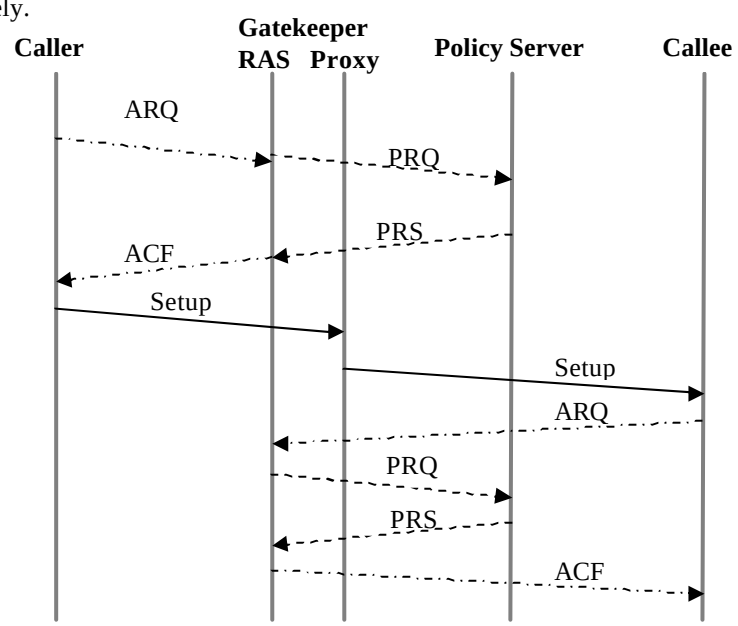

Figure 7. Admission Policy

3.3 Forward-Always Policy

The forward-always service is one of the basic supplementary services that the H.450 protocol defines. In H.450.3, forward-always permits incoming calls for a served user's number to be redirected to another number. Here, the callee is the served user's number. In H.450.3, the forward-always service can be enforced on the terminal, gatekeeper or proxy server. But in the H.323 policy architecture, we combine the gatekeeper and policy server. The H.323 policy confers two advantages. One advantage is that the forwarded-to terminal can be found once, unlike H.450.3 that must look for the forwarded-to terminal more than once. It is possible for the first or later forwarded-to terminal to select the forwarding service (forward-always or forward-on-busy). The other advantage is that our H.323 policy system can detect policy interactions, i.e. traditional feature interactions in call forwarding can be found.

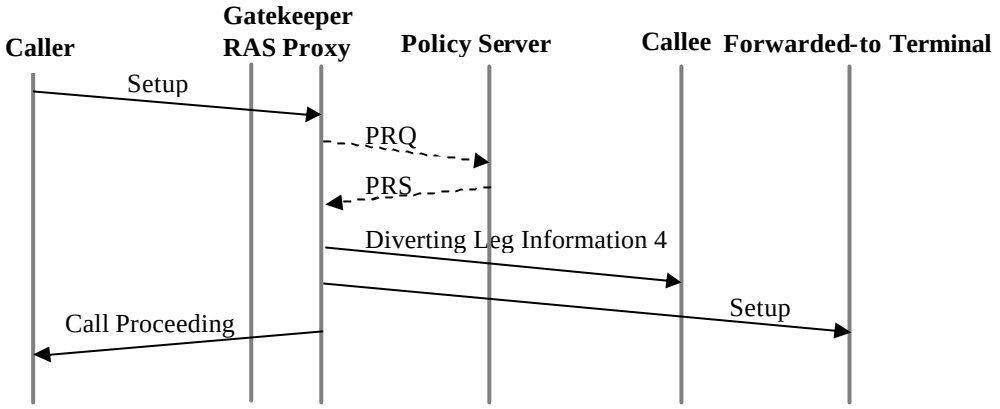

Figure 8. Forward-Always Policy

As Figure 8 shows, when the gatekeeper receives the *Setup* message from the caller, it creates a *PRQ* message from the call information in the *Setup* message. The information includes some necessary items and some optional items as shown in Table 2.

Information element	Status	Description
H323	Mandatory	Used to ask the policy server for H.323 policies
SERVER_NAME	Mandatory	Alias name of the gatekeeper
CallIdentifier	Mandatory	Globally unique call identifier; makes sure the policy will be applied to the corresponding call
Caller	Mandatory	Alias address of the source
Callee	Mandatory	Alias address of the destination
Trigger	Mandatory	The setup trigger; for "forward-always" this is the callee
User	Mandatory	Set as *Callee*
Other	Optional	If an element may be usable in a policy, it is passed to the policy server

Table 2. Fields of The PRQ Message for Forward-Always

After receiving the *PRQ* message, the policy server looks for related policies from the policy repository, and checks for policy conflict. Finally, the policy server responds to the gatekeeper's request with a series of actions. These could be *Continuing* or *Forwarding*. If the action is *Continuing*, the gatekeeper forwards the *Setup* message to the callee according to the normal call procedure. If the action is *Forwarding*, the *PRS* message contains more information such as a diversion number, diversion reason, and whether the callee is notified as listed in Table 3. In the *Forwarding* case, the gatekeeper sends the *Setup* message to the forwarded-to terminal. If *CalleeNotification* is true, the gatekeeper will send a *Diverting Leg Information 4* message to the callee using call-independent procedures to notify the callee that it has a call to be forwarded to the forwarded-to terminal.

Information Element	Status	Description
CallIdentifier	Mandatory	Not got from the policy repository but taken from the PRQ message; in a multi-threaded environment, it makes sure the policy is applied to the corresponding call
DiversionNumber	Mandatory	Number of the forwarded-to terminal
DiversionReason	Mandatory	Reason for forwarding
CalleeNotification	Optional	Indicates whether the gatekeeper needs to notify the callee

Table 3. Fields of The PRS Message for Forward-Always

If the caller, the callee and the forwarded-to terminal register with different gatekeepers, the other calling procedures, except communication with the policy server, are the same as for the description of the H.450 protocol; the details are not given here.

3.4 Forward-on-Busy Policy and Waiting-on-Busy Policy

An H.323 telephony system should provide service on busy. The H.450.3 specification describes the forward-on-busy service. This enables a callee to have calls addressed to a busy number to be redirected to another endpoint. The H.450.6 specification states that the waiting-on-busy service permits a busy callee to be informed of an incoming call while engaged with one or more other calls. Because both services are related a busy condition, we discuss them together.

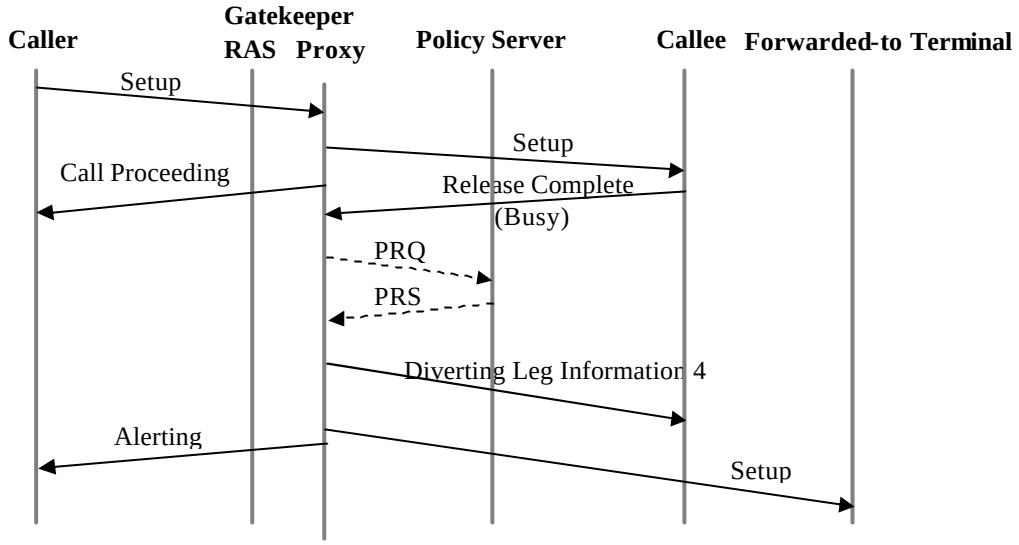

Figure 9. Forward-on-Busy Policy

After sending the *Setup* message to the callee, the gatekeeper receives a *Release Complete* message including the busy status from the callee. The busy event will trigger the corresponding policy. The gatekeeper extracts some information from the *Setup* message to create the *PRQ* message. The information fields are identical to the ones listed in Table 2 except for the *Trigger* item. The trigger should be "busy"; different events trigger different policies. The message flow is shown in 0.

The policy server will respond to the *PRQ* message with a *PRS* message including some actions. There are three types of actions. Firstly, the action may be *Continuing*. This means that the gatekeeper will send a *Release Complete* message to the caller according to normal calling procedures. Secondly, the action may be *Forwarding*. In this case, the fields of the *PRQ* message are identical to the ones listed in Table 3, except that the diversion reason is forward-on-busy. Thirdly, the action may be *Waiting*. In this case, the fields of the *PRQ* message are as shown in Table 4. In fact, this case is the waiting service. The call flow diagram is shown in 0. The gatekeeper will send an *Alerting* message including the information about waiting in order to let the caller keep ringing. At the same time, the gatekeeper will initiate a wait timer. On every *WaitingTimeInterval*, the gatekeeper will send a *Setup* message to the callee. If the gatekeeper has tried *TryTimes* times, the gatekeeper will send a *Release Complete* message to the caller to finish the call.

Information element	Status	Description
CallIdentifier	Mandatory	Same as Table 3
WaitingTimeInterval	Mandatory	Time interval to send the setup message to callee
TryTimes	Mandatory	Maximum times to try to establish a call

Table 4. Fields of The PRS Message for Waiting-on-Busy

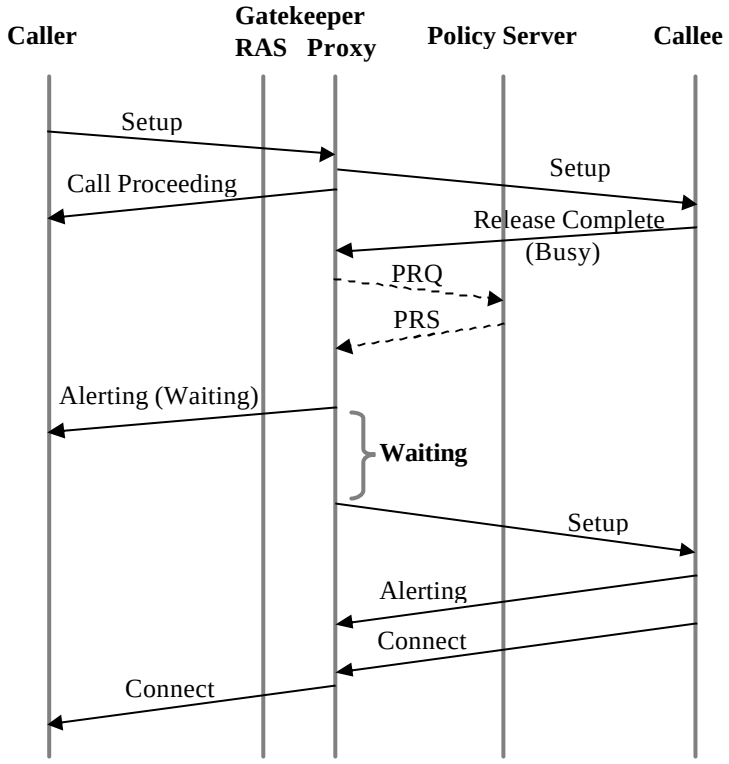

Figure 10. Waiting-on-Busy policy

3.5 Forward-no-Answer Policy

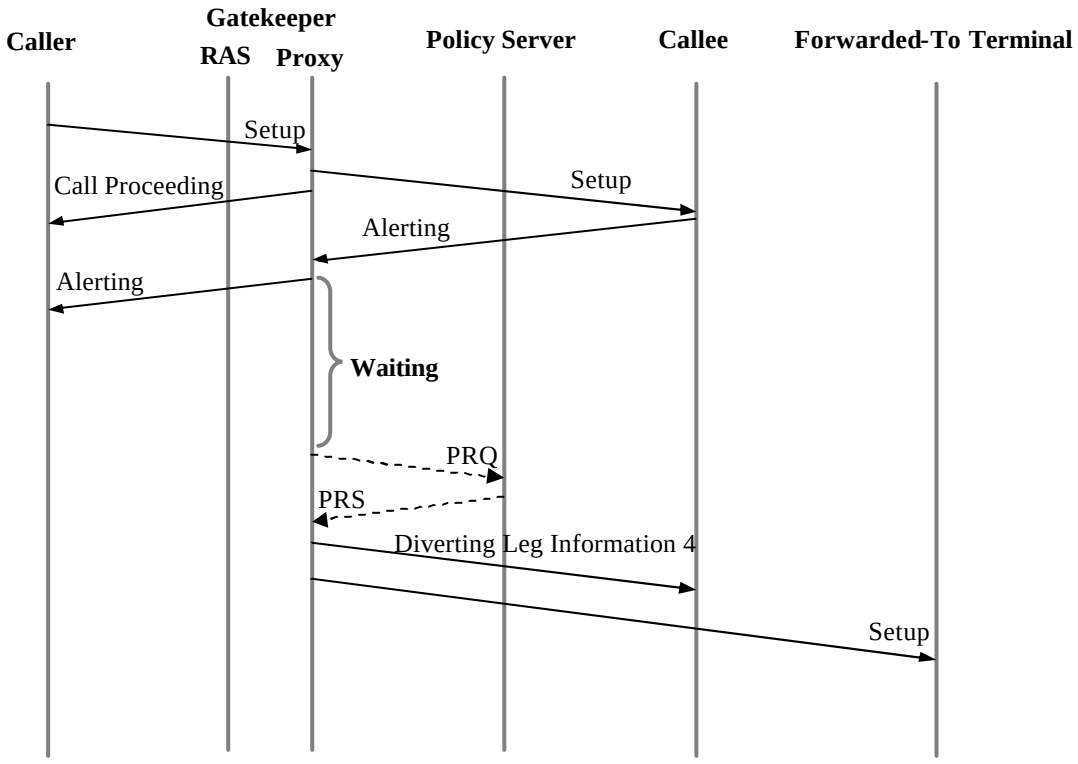

Figure 11. Forward-no-Answer Policy

For the forward-no-answer service, H.450.3 describes calls that are addressed to the callee's number but for which the connection is not established within a defined period of time. These are redirected to the forwarded-to terminal. Therefore, the trigger event for the policy is that the call cannot be established within a defined period of time; the trigger field is "no-answer". When the gatekeeper receives an *Alerting* message from the callee, it

sets one timer for this call. If the call is established soon, the gatekeeper will cancel the timer. Otherwise, the gatekeeper will extract some information from the *Setup* message to create a *PRQ* message. The information elements are the same as ones listed in Table 2 except that the trigger is forward-no-answer. The message flow is shown in 0.

The policy server will respond to the *PRQ* with two sorts of *PRS* message: *Continuing* or *Forwarding*. In the former case, the gatekeeper sends a *Release Complete* message to both the callee and the caller according to normal procedures. In the latter case, the gatekeeper firstly sends a *Diverting Leg Information 4* message to the called terminal. This message may either be sent within a *Facility* message on the existing call reference, or may be sent to the called terminal using call-independent procedures. Then the gatekeeper sends the *Setup* message to the forwarded-to terminal which is included in the *PRS* message.

3.6 Holding Policy

H.450.3 describes a call-holding service that allows a served user, the originally calling user (the caller) or the called user (the callee), to interrupt communications on an existing call. Subsequently, if desired, it re-establishes communication with the held user. There are several cases in the call-holding service. With regard to the holding location, either the source gatekeeper (the near end) or the destination gatekeeper (the remote end) can hold the call. As for the information to be provided to the held user by the gatekeeper during the call holding period, there are four choices:

- music/announcement in the audio logical channel
- video in the video logical channel
- video plus audio in the video and audio channels
- freeze-frame (still image) in the video channel, plus music/announcement in the audio channel.

How does H.323 actually work in practice? Without a policy, the served endpoint will firstly set the holding location manually. Then a *Facility (Holding)* message will be sent to the gatekeeper with the holding location element. The gatekeeper forwards the *Facility* message. Finally, the gatekeeper sends the media to the held endpoint according to the configuration defined in advance. In fact the served user cannot chose a preference for holding.

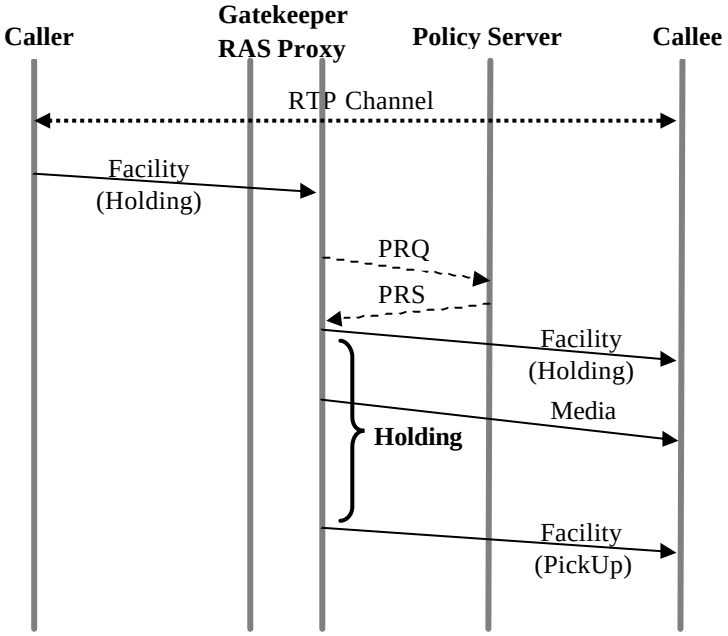

Figure 12. Holding Policy

We therefore introduce a policy to handle the call-holding service. During communication, when the gatekeeper receives a *Facility (Holding)* message, it creates a *PRQ* message from the *Facility* message. The *PRQ* message includes some necessary information elements listed in Table 3, but the trigger is "call-holding". The message flow is shown in 0.

The policy server responds to this situation. The *PRS* message may be either *Rejecting* or *Holding*. If it is *Rejecting*, the *PRS* message will give the reason. If it is *Holding*, the *PRS* message will include the call holding location and the media type for the held endpoint, as shown in Table 5.

Information Element	Status	Description
CallIdentifier	Mandatory	Same as Table 3
CallLocation	Mandatory	Either near-end or far-end call hold
MediaType	Mandatory	The media to provide to the held endpoint during holding

Table 5. Fields of The PRS Message for Call-Holding

3.7 Forking Policy

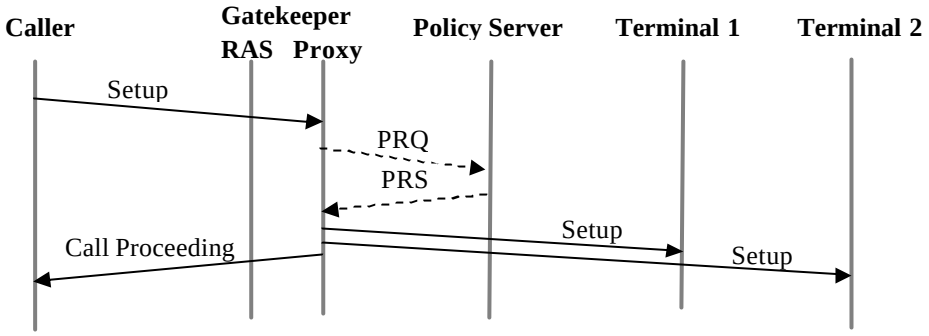

Figure 13. Forking Policy

A forking service is popular in traditional telecommunications systems, though H.323 does not support this explicitly. In general if a user would like to fork a call, the user must ask the administrator to set up a forking service in the gatekeeper manually; this is not a flexible solution. The message flow is shown in Figure 13.

If we introduce policies into the H.323 system, a forking service can be more flexible and can be set on the basis of the user's preference (e.g. forking to a different group of addresses at different times). When the gatekeeper receives a *Setup* message, it extracts some information elements listed in Table 3 to form a *PRQ* message. If the callee has set a policy for a forking service, the corresponding *PRS* message will contain the forked group and the enabled property, as shown in Table 6. The message flow is show in Figure 13.

Information Element	Status	Description
CallIdentifier	Mandatory	Same as Table 3
ForkingGroup	Mandatory	Group to which the call will be forked
Enabled	Mandatory	Indicates whether the forking service is enabled at the moment

Table 6. Fields of The PRS Message for Forking

The forking service interacts with the forward-always service. Likewise, there is a policy interaction between the forking policy and the forward-always policy. It is the responsibility of the policy server to resolve such policy interactions. The H.323 communication system does not therefore need to do anything about the interactions; the gatekeeper will receive a definitive resolution from the policy server. This is one advantage of using the policy server to resolve policy conflicts.

3.8 Call Intrusion Policy

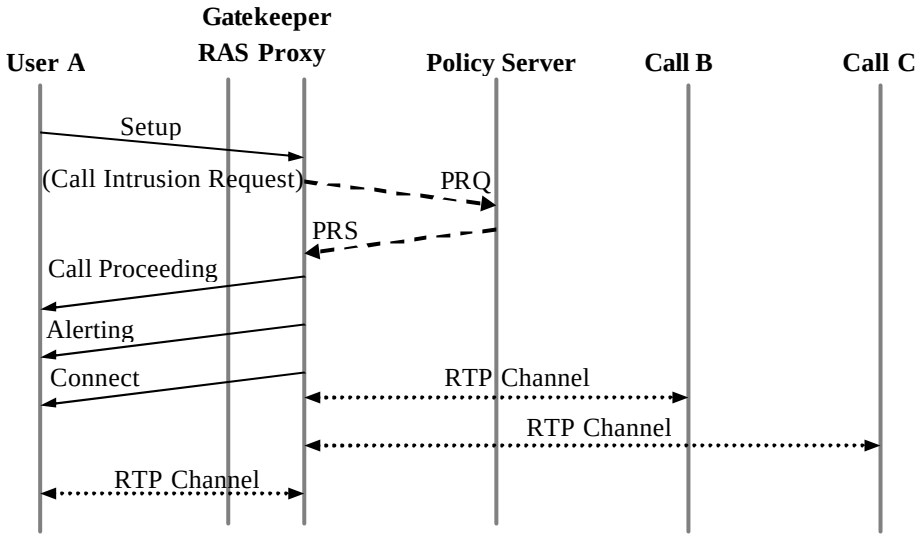

Figure 14. Intrusion Policy

The H.450.11 specification describes call intrusion as a service that, on request from the served user (A), enables the served user (A) to establish communication with a busy called user (B) by breaking into an established call between user B and a third user (C). As far as implementing the call intrusion service is concerned, there are several options:

- Conference type of connection: the served user (A), B and C are merged into a conference type of connection, that is they have an *ad hoc* conference.

- Held type of connection: the unwanted user C is split from B by B automatically invoking the call hold service.

- Silent monitoring type of connection: the served user (A) can just listen to (i.e. monitor) the established call.

- Forced release: the served user (A) requests forced release of the established call between B and C, so that A can then communicate with B.

- Wait on busy: the served user (A) may be able to request a transition from the intrusion state to a wait-on-busy state.

Because there are so many options to implement, it is difficult for an H.323 system to enforce call intrusion. The H.323 system should implement different options depending on the actual situation. So far, there is no H.323 system on the market to enforce the intrusion service with all options.

As 0 shows, this problem can be solved easily through use of policies. The user is able to set policies about call intrusion according to individual preferences. For example, the user can allow the boss to monitor calls. If there is very urgent call, call intrusion can select the option of forced release. If call intrusion is not vital, the gatekeeper can direct the served user (A) to a wait-on-busy state. The policy server should be responsible for filtering the most appropriate policy. In summary, the policy server provides definitive and accurate guidance to the gatekeeper through combining the current call intrusion information and the user's preferences.

When the gatekeeper receives a *Setup* message with *CallIntrusionRequest*, it will extract some information to create the *PRQ* message. Because the call intrusion service is more complicated than the forward-always service, more information elements are required as listed in Table 7. Notice here that the user is set as User B. This means the policy server will look up policies for User B.

Information Element	Status	Description
H323	Mandatory	Used to ask the policy server for H.323 policies
SERVER_NAME	Mandatory	Alias name of the gatekeeper.
CallIdentifier	Mandatory	Globally unique call identifier that makes sure the policy will be applied to the corresponding call
UserA	Mandatory	Alias address of the served user
UserB	Mandatory	Alias address of one terminal in the established call
UserC	Mandatory	Alias address of another terminal (the unwanted user) in the established call
Trigger	Mandatory	Set as "call-intrusion"
EmergencyCall	Mandatory	True or false
User	Mandatory	Set as User B
Other	Optional	If an element may be usable to the policy, it can be passed to the policy server

Table 7. Fields of The PRQ Message for Call Intrusion

The policy server will return the corresponding *PRS* message containing the information elements listed in Table 8. The element *NotifyOption* indicates whether the gatekeeper should notify the parties of the established call before call intrusion is implemented.

Information Element	Status	Description
CallIdentifier	Mandatory	Same as Table 3
EnforceOption	Mandatory	Type of the implementation
NotifyOption	Mandatory	True or false; it marks whether User B and User C are notified

Table 8. Fields of The PRS Message for Call Intrusion

3.9 Name Identification Policy

The H.450.8 specification describes a name identification service: the calling part name presentation/restriction is to provide/restrict the name of the calling party to the called party. In addition, the connected party name presentation/restriction is to provide/restrict the name of the connected party to the calling party. This definition is comprehensive. The name identification service can be enforced between two communicating parties, including gateways. Here we just consider the caller and the callee. In a traditional H.323 system, once the name identification configuration is set, the feature implementation is the same regardless of differing situations. On introduction of policies, this service becomes more flexible.

When will the gatekeeper connect to the policy server in order to look for policies concerning name identification? The trigger event is the gatekeeper receiving an *ARQ* message. When the source gatekeeper receives the *ARQ*, it asks for policies about the calling part name presentation/restriction. Likewise, when the destination gatekeeper receives an *ARQ*, it asks for policies about the called party name presentation/restriction. The call flow is that shown for admission in Figure 7.

The *PRQ* message contains the information elements listed in Table 2. The trigger ins this case is "Ingress-Setup" or "Egress-Setup". The *PRS* message is relatively simple, as listed in Table 9.

Information Element	Status	Description
CallIdentifier	Mandatory	Same as Table 3
Calling/called party name	Mandatory	Presentation/restriction

Table 9. Fields of The PRS Message for Name Identification

The source gatekeeper sends the calling party name information within an *H4501 Supplementary Service* message contained within the H.225.0 *Setup* message. The destination gatekeeper sends the called party name information within an *H4501 Supplementary Service* message contained within the H.225.0 *Connect* message.

3.10 Bandwidth Policy

Bandwidth management is an important function of a gatekeeper. If the gatekeeper is expected to implement precise bandwidth management, it needs to understand the network topology. The network has the responsibility

for QoS, such as RSVP and DiffServ, that is the network underlying routers can provide the bandwidth information to the gatekeeper. But currently many networks do not have QoS support so the gatekeeper must adopt a simple solution: the gatekeeper administrator sets a total amount of bandwidth, and the gatekeeper subtracts a certain amount for each call. For the Internet in general, this method works only roughly. However, for one PSTN gateway it is precise enough.

Throughout the call period, an endpoint, including the caller and the callee, can request a bandwidth change from its own gatekeeper (either increasing or reducing the bandwidth). The gatekeeper must also able to initialise a *BRQ* message according to the requirements. We discuss the different cases in the subsections below.

3.10.1 *ARQ* Message with Bandwidth Value

If an *ARQ* message requests bandwidth (if not, the bandwidth field is zero), the message flow for the bandwidth policy is as shown for registration in Figure 6. The caller and the callee request the bandwidth independently: through the use of H.245, they exchange their own capability sets. They transmit the media (audio and/or video) in a common format on the basis of this capability set.

Different policies can be set for bandwidth. A user may request different call quality according to the situation, including economic factors. An organisation may distribute the different amount of bandwidth on the basis of the user's status.

When the gatekeeper receives an *ARQ* message, it extracts the bandwidth request information to create a *PRQ* message. The bandwidth information is listed in Table 10.

Information Element	Status	Description
H323	Mandatory	Used to ask the policy server for H.323 policies
SERVER_NAME	Mandatory	Alias name of the gatekeeper
CallIdentifier	Mandatory	Globally unique call identifier that makes sure the policy will be applied to the corresponding call
Caller	Mandatory	Alias address of the source
Callee	Mandatory	Alias address of the destination
Trigger	Mandatory	Set as "Egress-Setup" (caller) or "Ingress-Setup" (callee); the permitted bandwidth for outbound and inbound directions may differ
Bandwidth	Mandatory	Requested bandwidth amount
ImportanceLevel	Mandatory	Importance level of this call
User	Mandatory	Set as *Caller* or *Callee*
Other	Optional	If an element may be usable to the policy, it can be passed to the policy server

Table 10. Fields of The PRQ Message for A Bandwidth Policy (1)

The policy server responds with two types of *PRS* messages. One type is *Bandwidth Rejecting* where the *PRS* message contains the reject reason field and the minimum bandwidth permitted. The other type is *Bandwidth Allowing* where the *PRS* message just confirms the request.

3.10.2 *BRQ* Message with Bandwidth Change

During the call, the caller or the callee might decide to use different media and need to change the bandwidth. A *BRQ* message will then be used. In addition, the gatekeeper may try to change the bandwidth with a *BRQ* message according to the busy/free situation. Firstly, we examine the different call flows concerning bandwidth policies: see Figure 16 and 0.

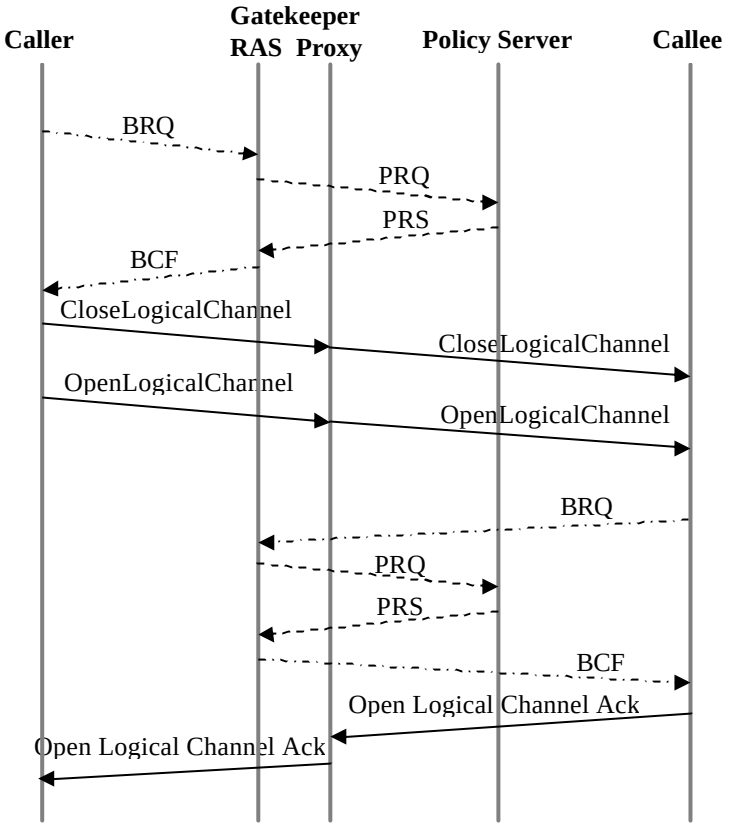

Figure 15. Bandwidth Policy: Media Transmitter changes The Bandwidth

The message flows for changing the bandwidth are different between the media transmitter and the media receiver. Here, we presume that the caller is the media transmitter and the callee is the media receiver. 0 shows how the media transmitter changes the bandwidth. If the caller wishes to increase its transmitted bit rate on a logical channel, it first determines if the call bandwidth will be exceeded. If it will, the caller must request a bandwidth change from its gatekeeper. The gatekeeper will communicate with the policy server. According to the reply of the policy server, the gatekeeper will send a *BCF* message to the caller. The caller sends a *Close Logical Channel* message to close the logical channel. It then reopens the logical channel using *Open Logical Channel* specifying the new bit rate. If the callee wishes to accept the channel with the new bit rate, it must first ensure that the call bandwidth is not exceeded by the change. If it is, the callee must request a call bandwidth change with its gatekeeper. When the call bandwidth is sufficient to support the channel, the endpoint replies with an *Open Logical Channel Ack*; otherwise the callee responds with an *Open Logical Channel Reject,* indicating an unacceptable bit rate.

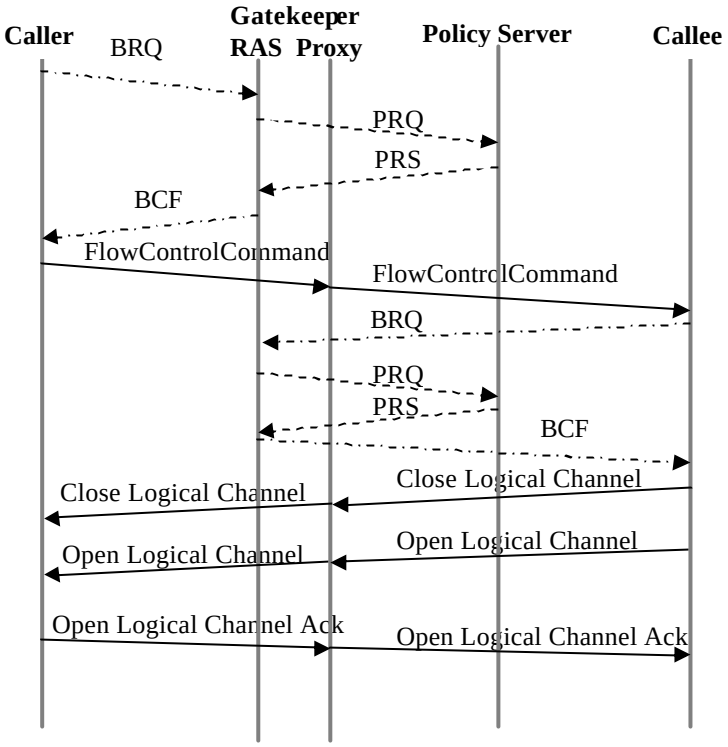

Figure 16. Bandwidth Policy: Media Receiver changes The Bandwidth

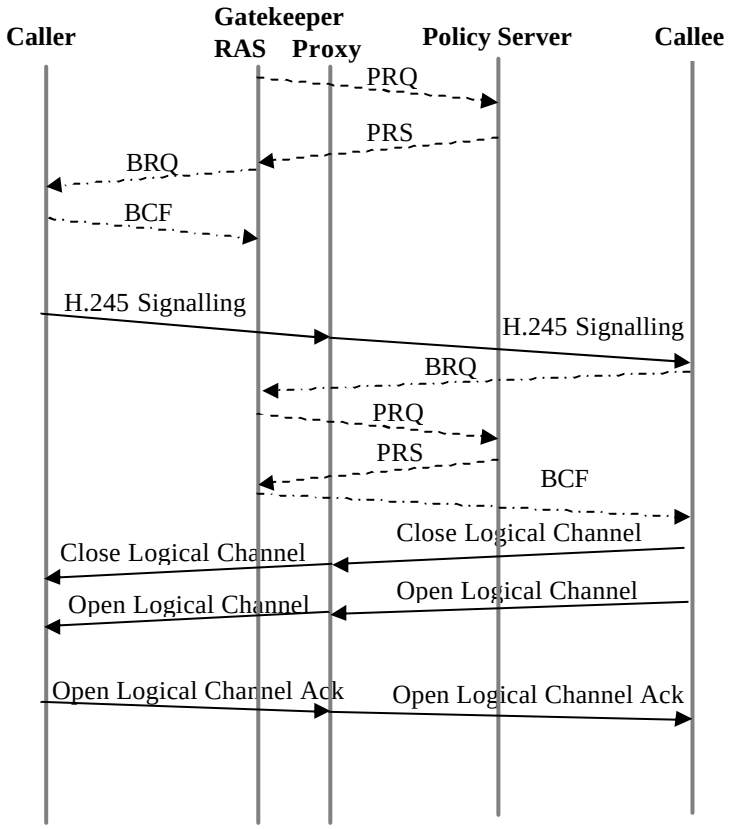

Figure 17. Bandwidth Policy: Gatekeeper requests A Bandwidth Change

Figure 16 shows how the media receiver changes the bandwidth. The difference from 0 arises after the caller gets permission from its own gatekeeper. A *Flow Control Command* instead is sent to the callee to indicate the new upper limit on the bit rate for the channel. Then the callee also requests a call bandwidth change with its

gatekeeper. After getting permission, the callee will initiate a *Close Logical Channel* message to close the logical channel, and then reopen it using *Open Logical Channel*.

Before sending the *BRQ* message to the caller, the gatekeeper will ask the policy server to look up the corresponding policy. On the basis of the policy server's response, the gatekeeper sends a *BRQ* message to the caller. If the caller can comply by changing the aggregate bit rate, it returns a *BCF* to the gatekeeper. It also sends the appropriate H.245 signalling message to inform the callee that bit rates have changed. The callee will request the change with its own gatekeeper. Likewise, the gatekeeper will communicate with the policy server. Finally, the bandwidth change has been effected.

Now we discuss which information fields the *PRQ* and *PRS* messages should contain. The gatekeeper will add the new bandwidth amount into the *PRQ* message. At the same time, it should still report the current bandwidth amount. The information fields of the *PRQ* message are listed in Table 11.

Information Element	**Status**	**Description**
H323	Mandatory	Used to ask the policy server for H.323 policies
SERVER_NAME	Mandatory	Alias name of the gatekeeper
CallIdentifier	Mandatory	Globally unique call identifier that makes sure the policy will be applied to the corresponding call
Caller	Mandatory	Alias address of the source
Callee	Mandatory	Alias address of the destination
Trigger	Mandatory	"Bandwidth Change"
NewBandwidth	Mandatory	Requested bandwidth amount
CurrentBandwidth	Mandatory	Current bandwidth amount
ImportanceLevel	Mandatory	Importance level of this call
User	Mandatory	Set as Callee or Caller
Other	Optional	If an element may be usable to the policy, it can be passed to the policy server

Table 11. Fields of The PRQ Message of Bandwidth Policy

Concerning *PRS* messages, there are two cases: *Rejecting* and *Permitting*. If the policy is to reject the bandwidth change, the *PRS* message will give the reject reason.

3.11 Summary of Supplementary Services

In this section, we have explored eleven policies related to H.323 supplementary services. From the foregoing, we have seen that there can be conflicts among some policies, such as between forward-always and waiting, since the services themselves may interact. In further work, we shall examine H.323 policy interactions and try to look for methods to resolve these.

4 Invocation of Policies on GNU Gk

Three open-source gatekeeper projects are known to the author:

- OpenGatekeeper: developed by Egoboo and available freely under MPL, this is a fully featured gatekeeper. Unfortunately, this project is inactive now; the author used to work on it previously.
- OpenGK: developed by Equivalence and available freely, OpenGK is rather simplistic.
- OpenH323 Gatekeeper [5]: developed by Citron Network, this project becomes increasingly more active, with more and more people involve in it. Its functionality is gradually increasing.

We chose the OpenH323 Gatekeeper. Because the names of these gatekeepers often confuse people, the original developers gave it another name, GNU Gk, which is used in this report.

4.1 Adapting GNU Gk

GNU Gk was developed with the basis of two underlying libraries. One is PWLib for I/O, multi-threading and so on. The other is the OpenH323Lib for implementing the H.323 protocol.

PWLib is the Portable Windows Library. It provides a method to produce applications that run on both Microsoft windows and Unix X-windows. It contains classes for I/O portability, multi-threading portability, aid in producing Unix daemons and NT services portably, and all sorts of Internet protocols.

OpenH323Lib is an open-source class library for the development of applications that use the H.323 protocol for multi-media communications over packet-based networks. It encapsulates a lot of H.323 entities including all kinds of H.323 signalling messages, the H.323 endpoint, the H.323 listener, the H.323 capability set, and so on.

GNU Gk consists of two main parts. One is the RAS Server/Client dealing with H.225 RAS messages. The other is the Proxy Server dealing with H.225 call signalling and H.245 control signalling. The two parts are separated, but are kept consistent through use of the same data such as the registration table, routing table and call table. The RAS server works on a fixed port: 1719. In general, the proxy server uses the default port 1720. If the proxy server needs to change the port, it can inform the endpoint by using H.225 RAS messages. Here, we explore this gatekeeper from the points of view of class inheritance, ER diagram, and operational procedures.

4.1.1 Class Inheritance

4.1.1.1 Thread Class Tree

Figure 18 displays the thread class tree of GNU Gk.

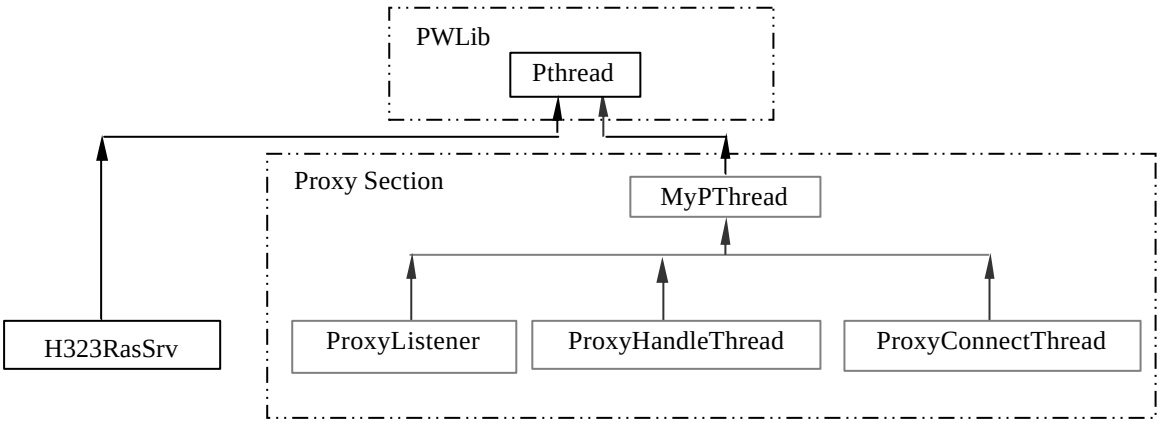

Figure 18. Thread Class Tree

- ProxyConnectTread: establishes a relationship between the local socket (caller–gatekeeper) and the remote socket (gatekeeper–callee).
- ProxyListener: listens to the H.225 call signalling interface.

- ProxyHandleThread: manages all the active sockets including CallSignalSocket, *H245Socket* and *UDPProxySocket*.

4.1.1.2 Socket Class Tree

Figure 19 lists the socket classes of GNU Gk that create network connections. In this figure, the classes enclosed by the dashed frame are provided by PWLib. PWLib provides different levels of classes operating I/O and sockets. GNU Gk inherits its special socket classes.

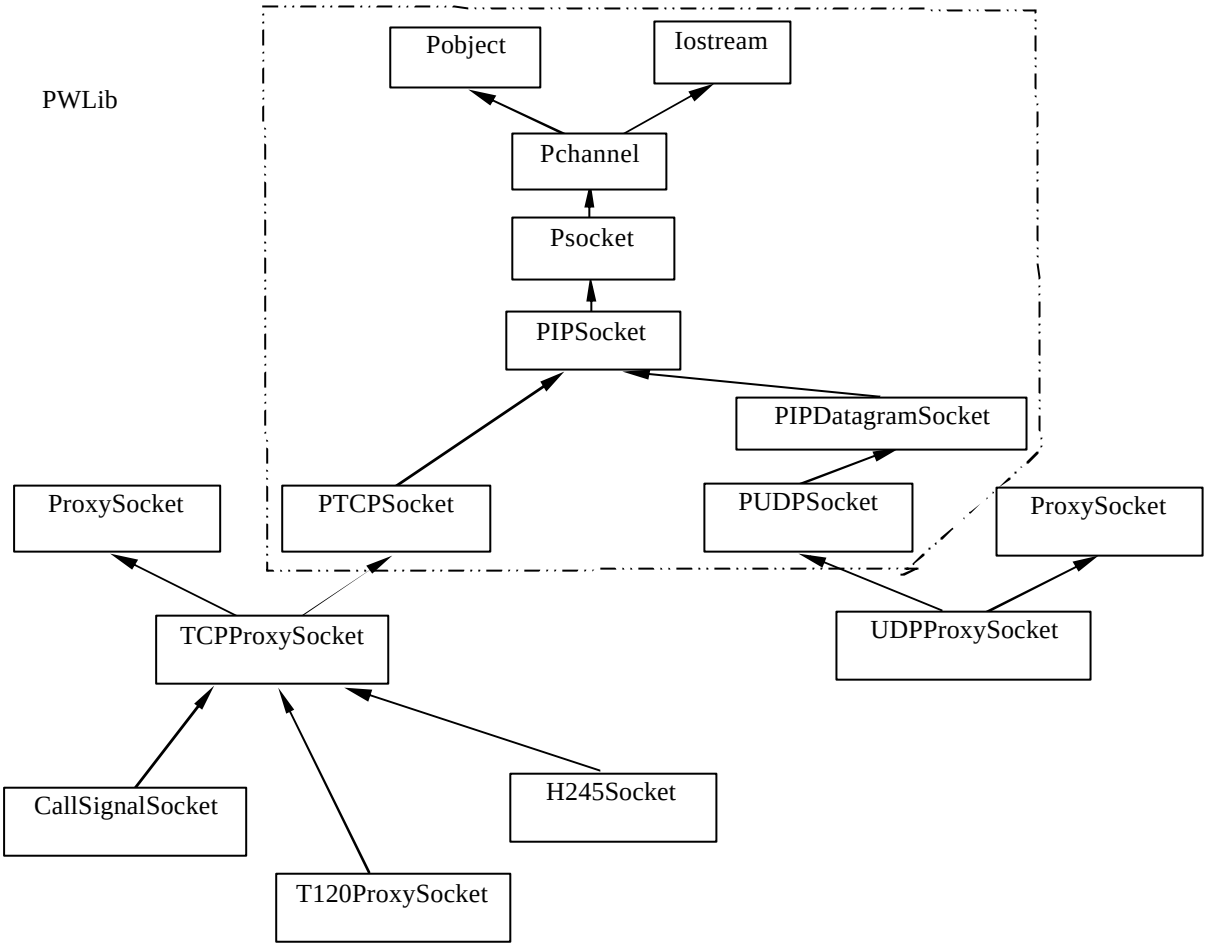

Figure 19. Socket Class Inheritance Tree

• ProxySocket: encapsulates the methods for receiving and transmitting data. Most methods are declared as virtual functions. It provides the socket interface for its descendant classes.

• TCPProxySocket: overrides the methods of the *ProxySocket* for receiving and transmitting data, and the methods of the *PTCPSocket* for listening to the network interface and connecting the socket.

• CallSignalSocket: a very important class that deals with H.225 call signalling messages such as *Setup* and *Connect*.

• H245Socket: handles an H.245 address, and exchanges the capability set.

• T120ProxySocket: overrides the *connectTo* method of *TCPProxySocket* and forwards the data.

• UDPProxySocket: establishes the UDP socket, the RTP and the RTCP channels.

4.1.2 ER Diagram

Besides the functions of a gatekeeper, GNU Gk can collect the call data for accounting and authorising the endpoint and the neighbour gatekeeper. Here, because a policy just changes the call routine, we mainly explore the RAS server/client (dealing with RAS messages), H.225 call signalling, and the H.245 call control section. Figure 20 and Figure 21 present the relevant ER diagrams.

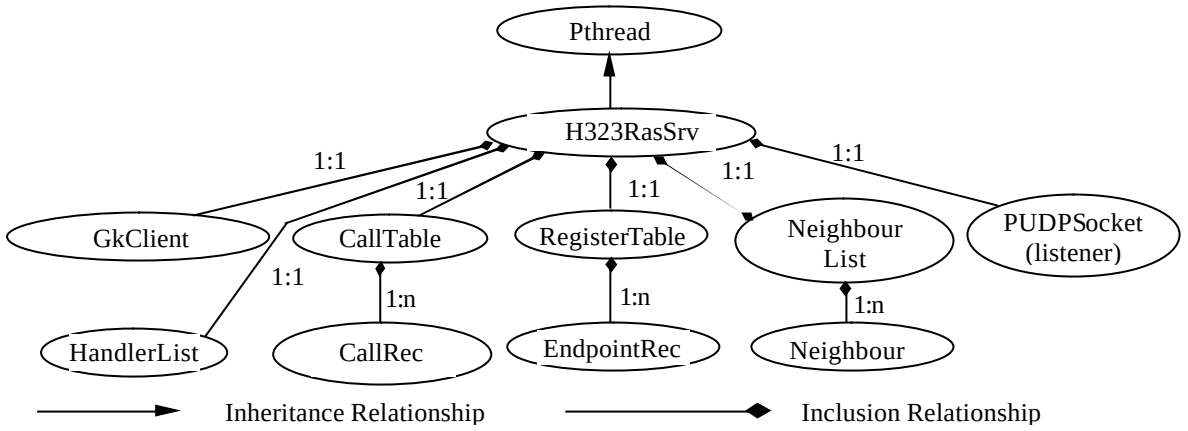

Figure 20. ER diagram of RAS Section

The RAS section handles all sorts of H.225 RAS messages. Dealing with RAS messages is a necessary aspect of a gatekeeper. The main class is *H323RasSrv*. This inherits from *Pthread* and makes its listener listen to the H.225 RAS network interface. Once an RAS message arrives, the listener will receive and dispatch it to the corresponding method. *H323RasSrv* deals with a *RegisterTable*. When *H323RasSrv* disposes of *LRQ/LCF, RRQ/RCF/RRJ* and *ARQ,* it will store or look up the endpoint information in the *RegisterTable*. *H323RasSrv* also maintains a *CallTable* to manage H.323 calls. When an *ARQ* or *BRQ* message arrives, *H323RasSrv* will work on the *CallTable*. Because GNU Gk supports cascaded gatekeepers and distributed gatekeepers, *H323RasSrv* maintains a *NeighbourList* for other gatekeepers. When *H323RasSrv* can not resolve an endpoint's address, it will ask for help from the gatekeepers listed by *Neighbourlist*. In addition, *H323RasSrv* starts the functions of *GK_Routed* and H.*245_Routed* though *HandlerList*. *HandlerList* is also used in a NAT (Network Address Translation) environment.

HandlerList maintains the entire proxy section. It includes one *ProxyListener* member listening to H.225 call signalling message, and at least two *ProxyHandlerThread* members managing active sockets and RTP/RTCP channels. GNU Gk can run on a multi-processor computer. The user can set the number of *ProxyHandlerThread* threads according to the running platform. Once a *Setup* message arrives, *ProxyListener* will create one *CallSignalSocket* object. This object is inserted into a queue maintained by *ProxyHandlerThread* to wait for service. GNU Gk uses *ProxyHandlerThread* to manage all the active sockets including *CallSignalSocket, H245Handler, H245ProxyHandler* and *RTPLogicalChannel* in order to make the software compact. *CallSignalSocket* deals with all H.225 call signalling messages, and does not die until the H.323 call has ended.

If the H.323 call works in H.245 mode, when *CallSignalSocket* disposes of the *Connect* message it will create one *H245Socket* object. This will exchange the capability set and create an *RTPLogicalChannel* object. If the H.323 call works in fast-start mode, when the *CallSignalSocket* deals with the *Setup* message it will exchange the capability set according to the *Setup* message and create an *RTPLogicalChannel*.

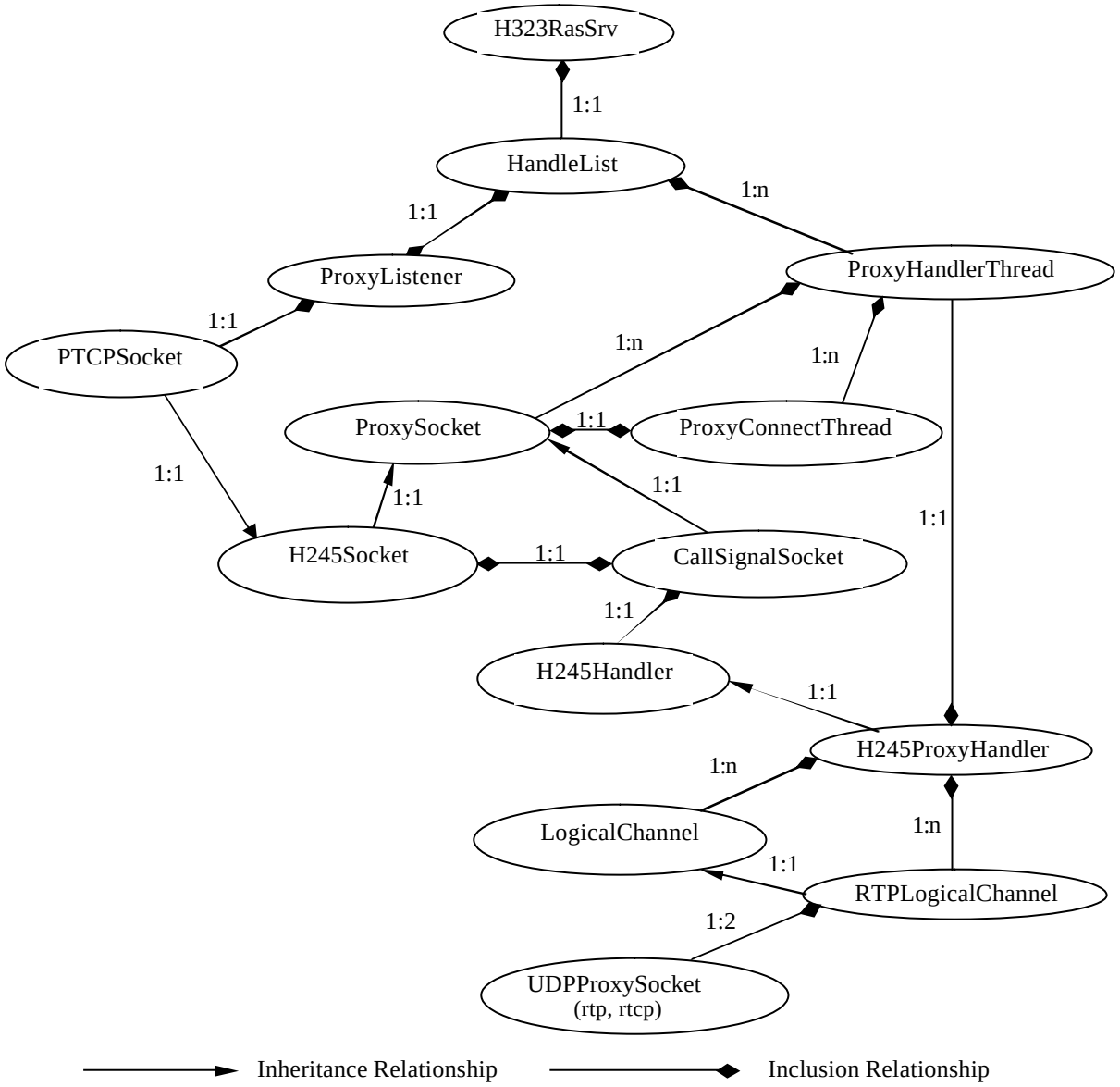

Figure 21. ER Diagram of Proxy Section

4.1.3 Call Procedure

As shown in Figure 22, there are four operational cases in H.323RasSrv. The first (e.g. *LRQ* message) is to deal with the RAS message locally without asking for help from other gatekeepers. The second (e.g. *ARQ* message) is to pass the RAS message to the other gatekeeper through the *GkClient* because the gatekeeper cannot deal with the message. The third is to forward the response message to the terminal; for example the gatekeeper may receive an *ACF* message and send it to the corresponding terminal. The fourth is to handle a request message from other gatekeepers, such as an *ARQ* message.

0 shows that the proxy section runs in four stages according to the H.323 protocol. The four stages are described in Section 2, so the description is not given here again.

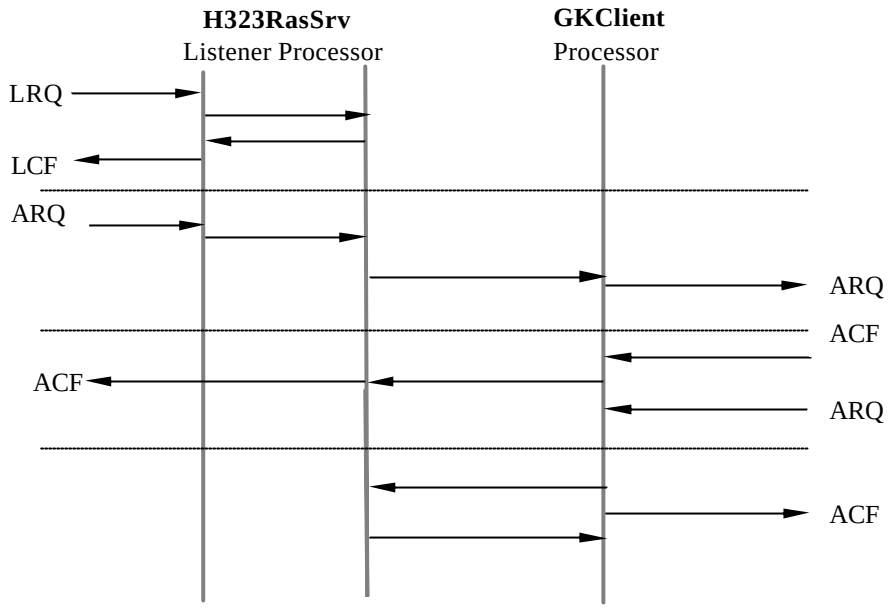

Figure 22. H323RasSrv Run-Time Diagram

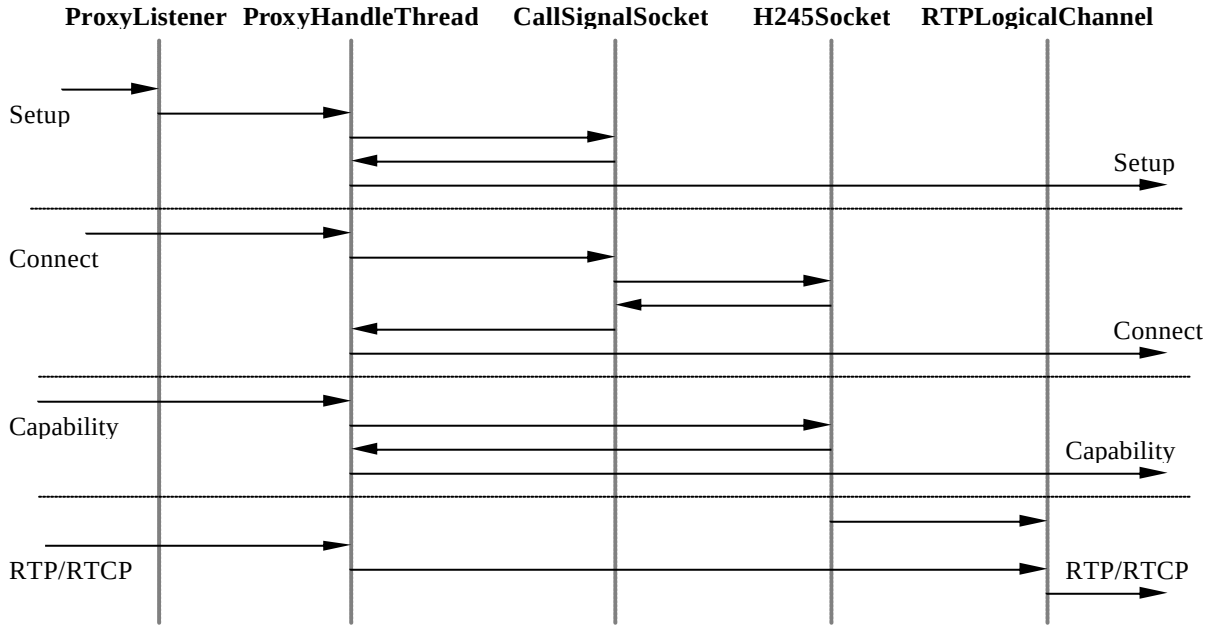

Figure 23. Proxy Section Runtime Diagram

4.2 Enforcing A Policy

Policy support has been developed, based on GNU Gk. In order to transplant policy functions to the new version of GNU Gk, we used a pre-processor and inheritance techniques. On the one hand, the new classes related to policies inherit from the present classes of GNU Gk. These inherited classes are put into independent files. For example the *PolicyCallSignalSocket* class inherits from the *CallSignalSocket* class, and the *PolicyH323RasSrv* class inherits from *H323RasSrv*. It is obvious that the characteristics of object-oriented programming are made use of. In places where the functions related to policies are invoked, the macro *POLICY* is used. Through defining this macro in the *Makefile* or not, we can decide whether policies are involved.

One aspect is the RAS Server/Client shown in 0. Because GNU Gk supports cascaded and distributed modes of working, it has both an RAS server and an RAS client. That is to say that two GNU Gks communicate in client-server mode.

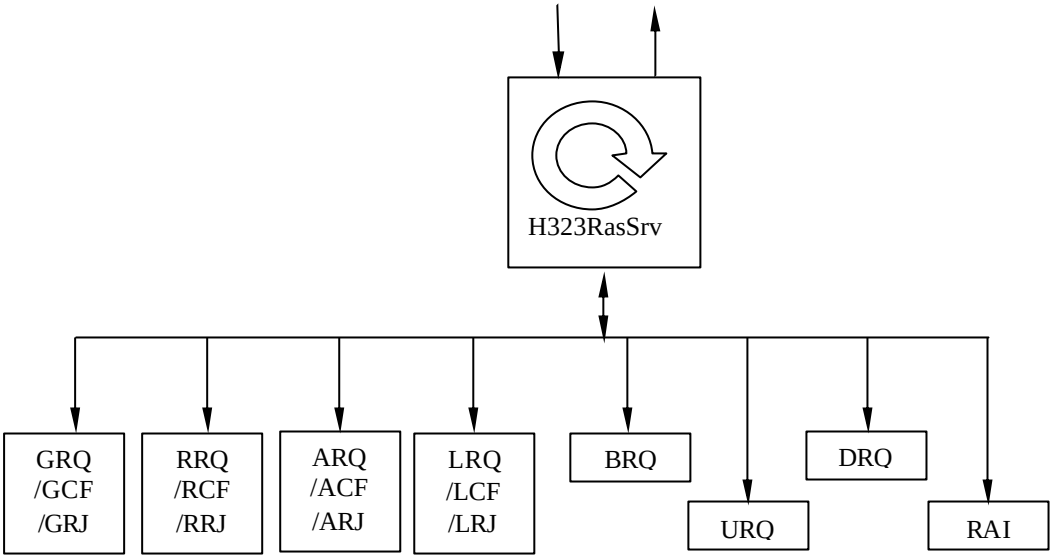

Figure 24. Modules of *H323RasSrv*

H323RasSrv is the main class. It runs a listening UDP socket, currently in a single thread. It is said that a forthcoming version will support multi-threading. According to the H.323 recommendation, RAS messages do not have a defined sequence and are relatively independent. Each member method of *H323RasSrv* deals with one RAS message. Some RAS messages are related to H.225 call signalling messages, such as the *ARQ* message.

As Figure 21 shows, the proxy section has a listening TCP socket called *ProxyListener*. When a new call is made, a new *callSignalSocket* object (TCP too) is created. It begins to listen to its own signalling interface. When a signalling message arrives, *CallSignalSocket* receives and dispatches it, and invokes the corresponding method. For example the *OnSetup* method is for the *Setup* message and the *OnConnect* method is for the *Connect* message. These call signalling messages occur in a strict order: the *Setup* message is the first; with *Alerting*, *Connect* and so on following. In fact, the proxy section has other components including *H245Socket* and *RTPLogicalChannel*. At the moment, policy-related services are related to just the *CallSignalSocket* class. So, here we give the details of only this class as shown in Figure 25.

Once policies are introduced, the general call flow has to be changed according to the actions required by the policy server. At the moment, a class named *PolicyCallSignalSocket* inherits from the *CallSignalSocket*. It adds a method to handle a policy, including extracting information from call messages, passing information to the policy server, and receiving the actions from the policy server. In addition it sets a timer. This is used to wait for the response of policy server. It is also used to trigger the corresponding policy, such as a forward-no-answer one. In addition *PolicyCallSignalSocket* needs to override some related methods for policies. For example, because many policies are related to a *Setup* message, it is necessary to override the *Setup* method. Moreover, the *Setup* message must be stored because it might be re-sent. The class adds one member variable named *m_setup* to record the *Setup* message. Likewise, *PolicyH323RasSrv* inherits from class *H323RasSrv* and overrides some corresponding methods. At the moment, *PolicyH323RasSrv* just overrides the member methods of *OnRRQ* and *OnARQ* to implement the registration policy and the admission policy discussed in Section 3. *PolicyCallSignalSocket* and *PolicyH323RasSrv* are shown in 0 and Figure 27.

Figure 28 gives the algorithm flow for the forward-no-answer policy.

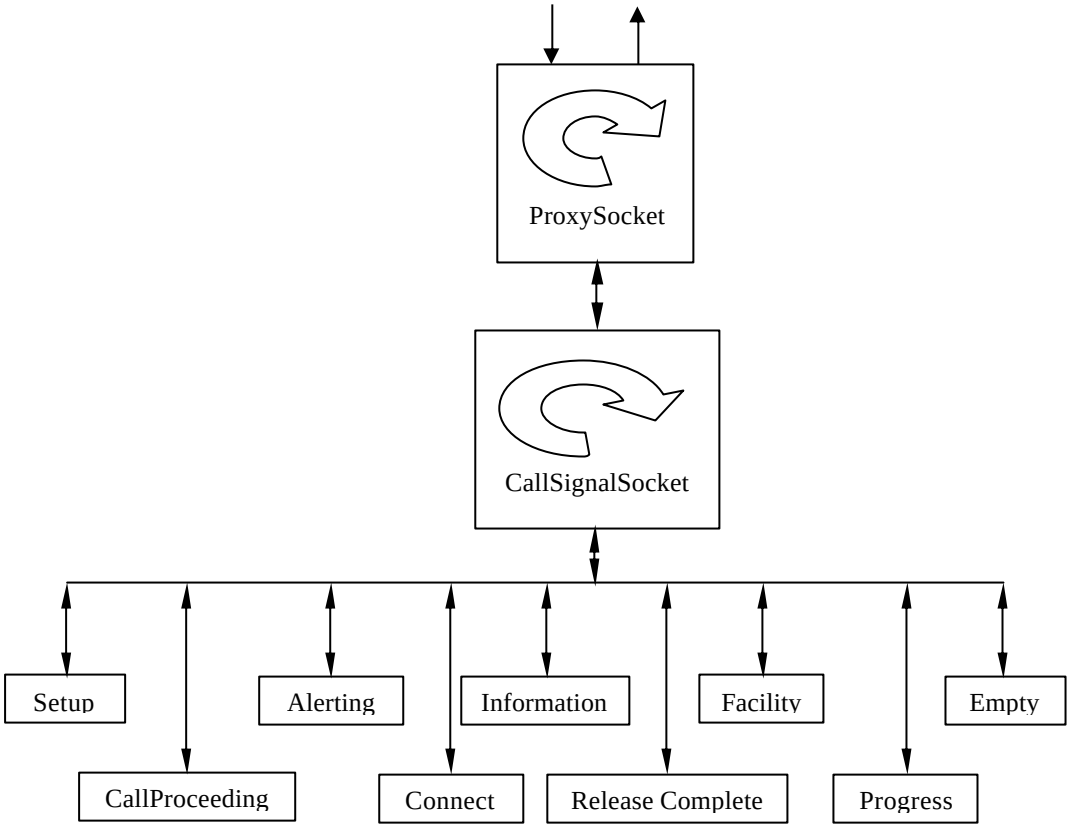

Figure 25. *CallSignalSocket* Class

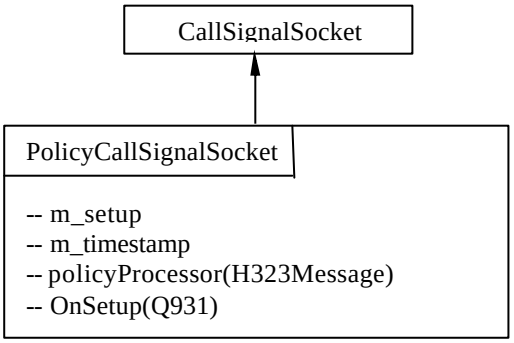

Figure 26. *PolicyCallSignalSocket* Class

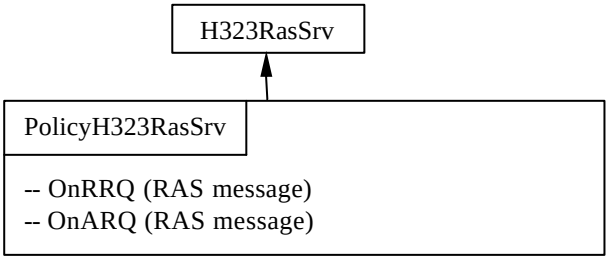

Figure 27. PolicyH323RasSrv Class

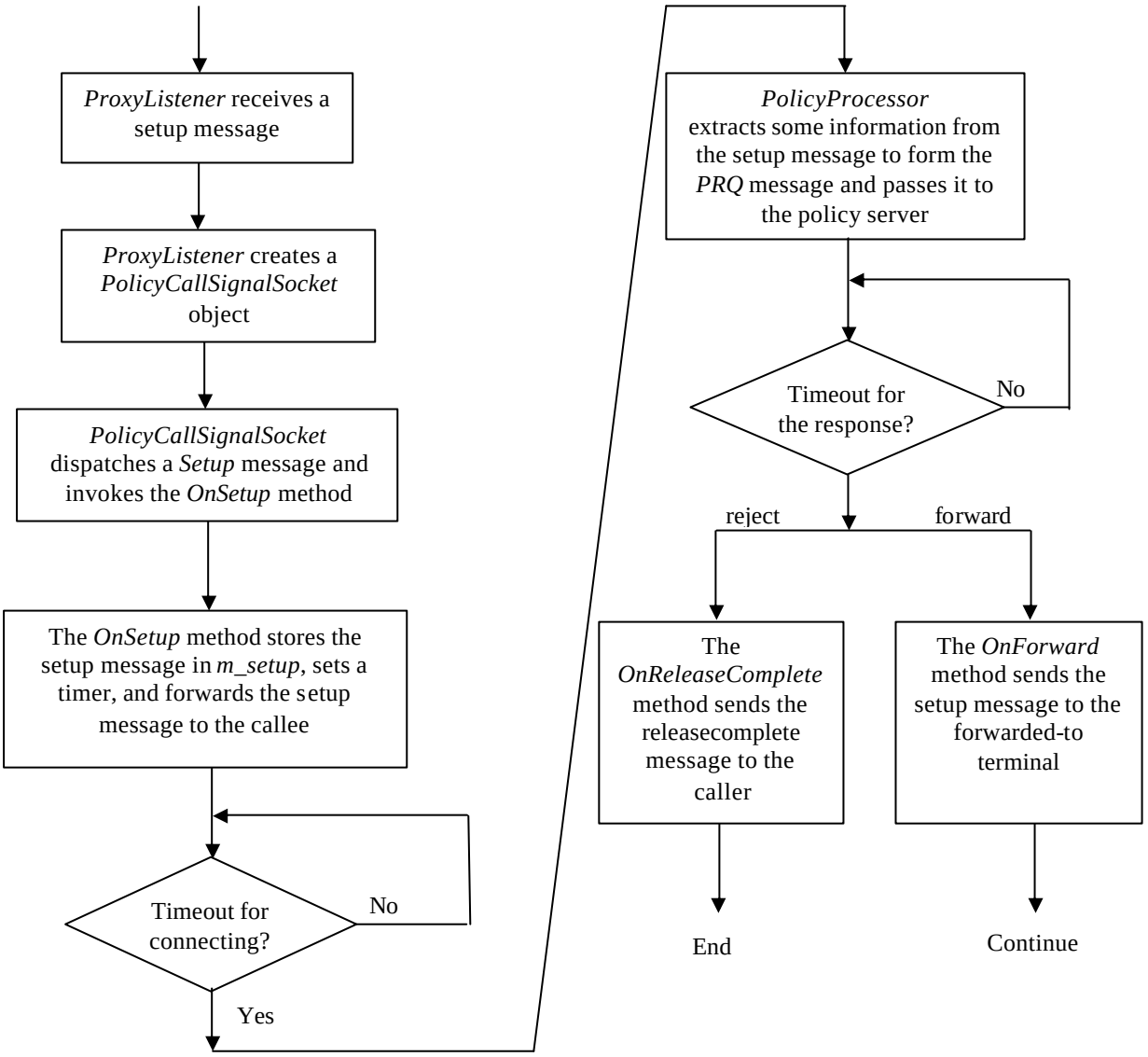

Figure 28. Enforcing The Forward-no-Answer Policy in GNU Gk

5 Designing H.323 Policies for The SIP Policy Server

5.1 Interface between The Gatekeeper and The Policy Server

The reference [9] has described the interface to the call control layer. To the H.323 telecommunications system, the call control layer is the gatekeeper. The communication architecture and the algorithm are identical with the description. Here, we shall discuss the specific points about H.323 policy interface.

5.1.1 From The Gatekeeper to The Policy Server

The gatekeeper monitors all H.323 calls. The policy module of the gatekeeper extracts information from H.323 messages and forms one structured string. Then the string is passed to the policy server. The policy module of gatekeeper suspends the call until a response comes from the policy server. For SIP policies, the structured string contains the current entire SIP message. But for H.323 policies, the structured string does not include any raw H.323 information. Before giving the reason, let us see how to pass an H.323 message into the H.323 system as shown in 0.

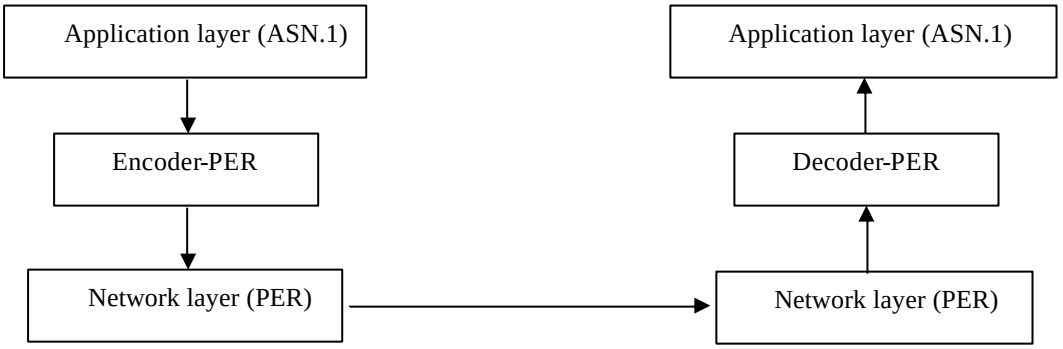

Figure 29. Formatting an H.323 Message

H.323 messages are stored in ASN.1 format in the application layer. The message structure primarily consists of an identifier and zero or more parameters. The identifier is used to identify the message, while the parameter fields convey the actual parameters. Each parameter also contains an identifying id and a content field. The content field supports a number of different data types, including raw, text, Unicode, *bool*, *number8*, *number16*, *number32*, *id*, compound and nested. This structure can define H.323 messages very flexibly. However, it is more difficult to understand the H.323 message compared to SIP messages, which are plain text. Before the H.323 message is passed via the network, it must be encoded in a binary format using packed encoding rules (PER). At the receiver, the H.323 message must be decoded into ASN.1 format. So an H.323 entity, such as a gatekeeper or a terminal, is required to include two parts. One is for analysing the H.323 message, the other is for encoding/decoding the H.323 message. At the moment, almost all the open H.323 projects are based on OpenH323Lib referred to in Section 3. OpenH323Lib is based on PWLib. In fact, both OpenH323Lib and PWLib are used at the same time.

Now we answer the question of why the body of an H.323 is not provided to the policy server. If the structured string passed from the gatekeeper to the policy server contains the raw H.323 message, the policy server must have the capability of decoding and understanding the H.323 message. In other words, the policy server is required to include the OpenH323Lib and the PWLib. There is no doubt this would considerably complicate the policy server. So, we choose a solution wherein the policy module of the gatekeeper extracts the information from the H323 message and puts it into a structured string. The information about an H.323 message is provided in (*variable*, *value*) pairs. In this way, the policy server gets a message that is plain text, as for support of SIP.

5.1.2 From The Policy Server to The Gatekeeper

Because H.323 messages have a different format from SIP, the interface from the policy server to the gatekeeper is H.323-specific. [9] gives details of the policy server interface.

For SIP policies, the interface passes serialised data including a number of actions. The actions have eight types based on the characteristics of SIP messages. At the beginning of the SIP call, the proxy server and the called party receive an INVITE message. During the call, other messages are constructed based on certain parameters in the INVITE message. The message has a relatively fixed structure that consists of the message header and the message body. In addition, a response message has a numeric response code. The interface is fairly straightforward.

However, it is impossible to characterise H.323 policies with a limited number of messages as is possible for SIP policies. From the discussion in Section 3, it can be seen that an H.323 message has dissimilar elements and a structure that is not fixed. Not only have the different messages different elements, but also the same kind of message created by different H.323 entities may have different elements. So the interface from the policy server to the gatekeeper uses the structured string made of many variable-value pairs. The policy module of the gatekeeper enforces a policy through combining the response from the policy server and the call context. This solution makes the response string from the policy server very long. But the approach is usable.

5.2 H.323 Protocol and Policy Terminology Mapping

The policy server is generic, that is it serves for several sorts of telecommunications systems including H.323, SIP and PBX. Different protocol terms must therefore be translated to matching policy terms. The policy server sets up a mySQL database with two tables that provide information about the mappings between the policy terminology and the respective meaning in protocol terms. [9] gives a detailed description of this approach. Mappings for the H.323 protocol terminology are given in Table 12.

Id	Protocol	Protocol Term	Policy Term	IsAction
1	H323	incoming_setup	incoming	no
2	H323	outgoing_setup	outgoing	no
3	H323	setup	all	no
4	H323	release complete	call termination	no
5	H323	connect	call connect	no
6	H323	forward_to(arg1)	forward_to(arg1)	yes
7	H323	block_call	block_call	yes
8	H323	BRS:arg1	bandwidth(arg1)	yes
9	H323	BRQ:arg1	request_bandwidth(arg1)	no
10	H323	no_answer	no-answer	no

Table 12. H.323 Terminology Mapping

5.3 H.323 Policies

The policy server is made of three part shown in Figure 30: a database, tuple space server and policy engine. The database provides information about the mapping between the policy terminologies, and their respective meaning in protocol terminology. Because the policy server needs to communicate with several different underlying communication layers such as SIP and H.323, it is necessary to translate protocol terms into the common policy terms. The tuple space server stores all policies, providing an XML access interface. Policies can be retrieved and stored in the special policy language. The policy engine is the most important part, being both complex and powerful. Firstly, an interface is provided to communicate with the underlying communication layers. Then, all policies related to the user are retrieved. After retrieval, the policy engine filters the retrieved policies according to the call environment variables. After filtering, the policy engine detects and resolves policy interactions. Finally, the policy engine responds to the underlying communications layer with the final actions of the policies.

Most components of the policy server are common for all the underlying communications layers. For H.323 policies, two parts of the policy engine need special support. First, the communications interface with the H.323 gatekeeper is special. Because an H.323 call message is transmitted in ASN.1-encoded format, ASN.1 is deliberately hidden from the policy server. The gatekeeper therefore extracts all call information and passes it in a simple and long structured string. Likewise, the policy server responds with final actions in a structured string. An example appears below (with \n denoting a newline):

H323\n
SERVER_NAME:gatekeeper1@stir.ac.uk\n
Trigger:IN-ARQ\n
Caller:2000\n
Callee:2002\n
MSG\n\n

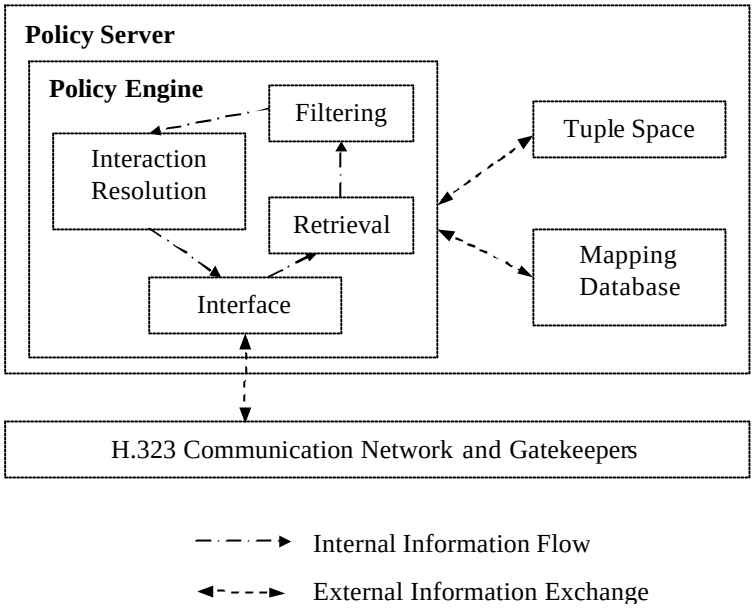

Figure 30. Components of The Policy Server

This is a structured-string passed from a gatekeeper to a policy server. The protocol identifier is at the beginning of this message. The call information variables are listed, including *SERVER_NAME*, *Trigger*, *Caller* and *Callee*. The call message may be provided at the end of these. As discussed above, the H.323 interface string does not involve the H.323 call message. But in order to be compatible with other protocols, a blank message body *MSG* is provided.

The protocol handler module also needs to be adapted for .323. The policy engine does not analyse H.323 call messages. The protocol handler module for H.323 finishes the translation of call environment variables from H.323 terminology to policy terminology, and the translation of the final actions from policy terminology to H.323 terminology.

In the mapping database, some items about H.323 terminology are added. If some H.323 terminology is different from policy terminology, it is necessary to list this in the terminology mapping table. Table 12 lists some examples. The *Id* field is the primary key of the table. The *Protocol* indicates H.323. The term "incoming" (row 1) is policy terminology, corresponding to "incoming_setup" in H.323 The field of IsAction is used to mark the translation direction. The value "no" means that this is call information not an action.

5.3.1 Forward-No-Answer policy

The forward-no-answer service is very popular. Likewise, the forward-no-answer policy is often used in the H.323 telephony system. Here, we discuss how to describe it using the policy language.

Suppose a user's number is "2000". The user defines three policy rules for the forward-no-answer policy. Their trigger event is "no-answer". Based on the call information, the rules are checked in the specified order. The policy engine first determines whether the first rule is applicable. If it is then the policy engine applies it, otherwise, the second rule is checked.

The first rule says that when no-answer happens, if the caller is "2001" or "2002" then the telephone call is forwarded to (a mobile) phone number "5000".

The second rule says that when no-answer happens, if I am busy, the time is 09.00–10.00 or 14.00–15.00, the call type is point-to-point, and the call topic is Java, the call is forwarded to a Java expert.

The third rule says that when no-answer happens, the telephone call is forwarded to my voice mailbox: 2000@cs.stir.ac.uk.

The policy is described in the simple policy language as follows. Note the XML has been simplified by omission of the trailing tags.

```
<policy owner="me@company.com" appliesTo="2000" id="no_answer" enabled="true">
  <policyrules>                                    <!-- all policy rules -->
    <sequential/>                                  <!-- try rules in sequence -->
      <polrules>                                   <!-- policy rules group -->
        <policyrule>                               <!-- policy rule 1 -->
          <triggers>                               <!-- triggers group -->
            <trigger>                              <!-- trigger 1 -->
                <trigger_name>no_answer            <!-- no answer? -->
```

```
  <conditions>                                  <!-- conditions group -->
   <conds>                                      <!-- conditions -->
        <condition>                             <!-- condition 1 -->
         <param>caller                          <!-- caller is 2001, 2002? -->
         < compop>in
         <value>[2001,2002]
  <actions>                                     <!-- actions group -->
   <acts>                                       <!-- actions -->
        <action arg1="5000">                    <!-- forward to 5000 -->
         <action_name>forward_to(arg1)
<polrules>                                      <!-- policy rules group -->
 <policyrule>                                   <!-- policy rule 2 -->
  <triggers>                                    <!-- triggers group -->
   <trigger>                                    <!-- trigger 2 -->
        <trigger_name>no_answer                 <!-- no answer? -->
   <conditions>                                 <!-- conditions group -->
        <and/>                                  <!-- both conditions hold? -->
         <conds>                                <!-- conditions -->
          <condition>                           <!-- condition 2 -->
           <param>busy
           <compop>is                           <!-- callee is busy? -->
           <value>true
         <conditions>                           <!-- conditions group -->
          <and/>                                <!-- conditions 3 or 4, 5, 6 hold? -->
           <conds>                              <!-- conditions -->
                <or/>                           <!-- condition 3 or 4 holds? -->
                 <conds>                        <!-- conditions -->
                  <condition>                   <!-- condition 3 -->
                   <param>time
                   <compop>in                   <!-- time is 09.00 to 10.00? -->
                   <value>[0900,1000]
                 <conds>                        <!-- conditions -->
                  <condition>                   <!-- condition 4 -->
                   <param>time>
                   <compop>in                   <!-- time is 14.00 to 15.00? --
                   <value>[1400,1500]
           <conditions>                         <!-- conditions group -->
                <and/>                          <!-- conditions 5, 6 hold? -->
                 <conds>                        <!-- conditions -->
                  <condition>                   <!-- condition 5 -->
                   <param>call_type
                   <compop>is                   <!-- call is point-to-point? -->
                   <value>point_to_point
                 <conds>                        <!-- conditions -->
                  <condition>                   <!-- condition 6 -->
                   <param>topic
                   <compop>is                   <!-- topic is Java? -->
                   <value>Java
   <actions>                                    <!-- actions group -->
        <acts>                                  <!-- actions -->
         <action arg1="Java expert">            <!-- forward to Java expert -->
          <action_name>forward_to(arg1)
<polrules>                                      <!-- policy rules group -->
 <policyrule>                                   <!-- policy rule 3 -->
  <triggers>                                    <!-- triggers group -->
   <trigger>                                    <!-- trigger -->
        <trigger_name>no_answer                 <!-- no answer? -->
  <actions>                                     <!-- actions group -->
   <acts>                                       <!-- actions -->
        <action arg1="2000@company.com">        <!-- forward to voicemail -->
```

<action_name>forward_to(arg1)

5.3.2 Bandwidth Policy

A bandwidth request is a special request message for an H.323 call. Therefore, a bandwidth policy is defined for the H.323 underlying communication layer. The management device of an H.323 network, the gatekeeper, can manage the bandwidth. Bandwidth is an aspect of QoS (Quality of Service). When the caller and the callee request admission from the gatekeeper, the admission request message (*ARQ*) might contain a bandwidth request message. During the call, the caller, the callee or the gatekeeper can ask to change the bandwidth amount. The user gains the best economical calls through setting the bandwidth policy. The organisation obtains good use of the network bandwidth though bandwidth policies.

Here, we discuss an example. The bandwidth policy is made up of two policy rules, checked in the specified order. They are described in the policy language.

The first rule says that when a bandwidth request happens, if the requester is from an H.323 network, traffic load is high, there is no emergency, priority is normal, and bandwidth request is for more than 500K, the action is rejection with two arguments. One is the reason (bandwidth lack) and the other is the allowed maximum bandwidth (100K).

The second rule says that when a bandwidth request happens, if there is an emergency or the priority is high, the bandwidth is granted.

```
<policy owner="me@company.com" appliesTo="2000" id="bandwidth_request" enabled="true">
 <policyrules>                                  <!-- all policy rules -->
  <sequential/>                                 <!-- try rules in sequence -->
   <polrules>                                   <!-- policy rules group -->
    <policyrule>                                <!-- policy rule 1 -->
     <triggers>                                 <!-- triggers group -->
      <trigger>                                 <!-- trigger 1 -->
       <trigger_name>bandwidth_request          <!-- bandwidth request? -->
     <conditions>                               <!-- conditions group -->
      <and/>                                    <!-- conditions 1 to 5 hold? -->
       <conds>                                  <!-- conditions -->
            <condition>                         <!-- condition 1 -->
             <param>network_type
             <compop>is                         <!-- network is H.323? -->
             <value>H323 Network
            <conditions>                        <!-- conditions group -->
             <and/>                             <!-- conditions 2 to 5 hold? -->
              <conds>                           <!-- conditions -->
               <condition>                      <!-- condition 2 -->
                    <param>traffic_load
                    <compop>is                  <!-- traffic load is high? -->
                    <value>high
              <conditions>                      <!-- conditions group -->
               <and/>                           <!-- conditions 3 to 5 hold? -->
                    <conds>                     <!-- conditions -->
                     <condition>                <!-- condition 3 -->
                      <param>emergency
                      <compop>is                <!-- not emergency? -->
                      <value>no
                    <conditions>                <!-- conditions group -->
                     <and/>                     <!-- conditions 4 to 5 hold? -->
                      <conds>                   <!-- conditions -->
                       <condition>              <!-- condition 4 -->
                            <param>priority
                            <compop>is          <!-- normal priority? -->
                            <value>normal
                      <conds>                   <!-- conditions -->
                       <condition>              <!-- condition 5 -->
                            <param>bandwidth
                            <compop>gt          <!-- bandwidth over 500 Kbps? -->
                            <value>500K
```

```
  <actions>                                          <!-- actions group -->
   <acts>                                            <!-- actions -->
    <action arg1="bandwidth-lack" arg2="100K">       <!-- reject request -->
        <action_name>bandwidth_reject(arg1, arg2)
<polrules>                                           <!-- policy rules group -->
 <policyrule>                                        <!-- policy rule 2 -->
  <triggers>                                         <!-- triggers group -->
   <trigger>                                         <!-- trigger 2 -->
    <trigger_name>bandwidth_request                  <!-- bandwidth request? -->
  <conditions>                                       <!-- conditions group -->
   <conds>                                           <!-- conditions -->
    <condition>                                      <!-- condition 6 -->
        <param>emergency
        <compop>is                                   <!-- is emergency? -->
        <value>yes
  <actions>                                          <!-- actions group -->
   <acts>                                            <!-- actions -->
    <action>                                         <!-- confirm bandwidth -->
        <action_name>bandwidth_confirm
```

5.4 H.323 Policy Parts of Policy Server

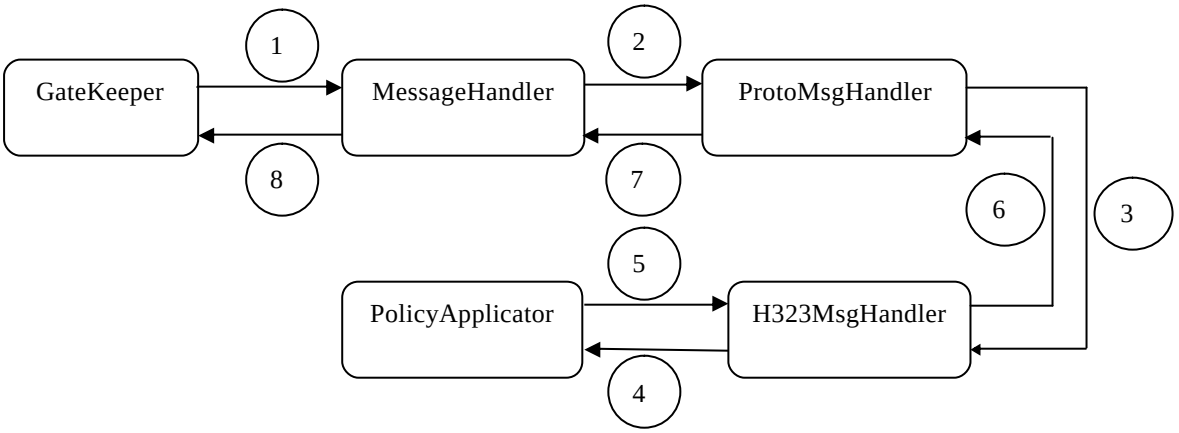

Figure 31. H.323 Policy Server

Figure 31 shows the implementation flow for H.323 polices. Most of the policy server classes are defined in [9], but some are special for H.323. One is the *H323MsgHandler* class, while another is some filtering class.

5.4.1 The Handler of An H.323 Message: *H323MsgHandler*

H323MsgHandler is a special class for H.323 messages. It inherits from the *GenericProtocolHandler* class. This is an abstract class with two member functions: *initialise* and *handleMsg* that are implemented by *H323MsgHandler*.

Similar to *SIPMsgHandler*, the method *initialise* initialises an *H323MSgHandler* object. It sets the policy store variable of an *H323MsgHandle,r* and also retrieves the action and event condition hashes from the policy store.

The method *handleMsg* deals with three main tasks. The first is responsible for analysing the H.323 message. As mentioned previously, unlike SIP messages an H.323 message is not sent to the policy server. A *PRQ* message mainly consists of some variable-value pairs. So, *handleMsg* places these environment variables into a hashtable from the PRQ message. The second task is to invoke *PolicyApplicator.* The third task is to form the *PRS* message (a structured string) from the actions returned by *PolicyApplicator.*

As shown in Figure 31, *H323MsgHandler* is invoked by *ProtoMsgHandler*. *ProtoMsgHandler* invokes a specific class, such as *SIPMsgHandler* or *H323MsgHandler*, according to the protocol header of the incoming message. *H323MsgHandler* then invokes *PolicyApplicator* in order to retrieve, filter and resolve polices.

5.4.2 The Filtering Classes for H.323 Environment Variables

The policy server has a package for filtering call environment variables. The *PolicyApplicator* class invokes this in order to finish filtering polices. In the package, each environment variable has its own filtering class. For example, the environment variable *caller* has filtering class *callerEnvEvaluate*. Some environment variables such as *caller* and *callee* have the same meaning in all underlying communications systems. However, some environment variables are for specific communications networks. For example, *topic* is used by the SIP protocol and *bandwidth* is used by the H.323 protocol. So, some special filtering dasses need to be added for H.323 policies. For the moment, the following ones have been added. In future, if necessary, some new filtering classes could be added.

All filtering classes inherit from an abstract class *EnvEvaluate*. This defines some ordinary comparison operations. These may not make sense for all environment variables. For example, for a Boolean variable the operators *gt* and *lt* may not be very meaningful. However, all filtering classes need to implement all comparison operations. For those operations that are not applicable, their implementation body just returns false. In addition, some individual filtering class may add special comparison operations.

eq: equal
ne: not equal
ge: greater than or equal to
gt: greater than
in: inclusion
nin: not inclusion ('not in')
le: less than or equal to
lt: less than

Some special filtering variable classes for the H.323 protocol are listed below:

activeMCEEnvEvaluate: filters boolean *activeMCE*, marking whether an active MCE (Multipoint Control Entity) is involved. This occurs with *Setup* and *ARQ*.

bandwidthEnvEvaluate: filters numeric *bandwidth*, standing for the amount of the resource bandwidth. It occurs with *ARQ* and *BRQ*.

capabilitySetEnvEvaluate: filters string *capabilitySet*, representing the capability set of the H.323 endpoints. It is contained in an H.245 message. Notice the difference between this and capability, which is described in [9].

srcCallSignalAddressEnvEvaluate, **destCallSignalAddressEnvEvaluate**, **signallingIP**: filters *sourceCallSignalAddress*, *destCallSignalAddress* and *callSignalAddress* respectively. These environment variables often occur in H.323 messages.

conferenceTypeEnvEvaluate: filters *callType*. Notice that here *callType* has a different meaning from the one defined for SIP in [9].

5.4.2 An Example: Forward-Always Policy

In order to show how an H.323 message passes through the implementation flow, an example (forward-always policy) is given.

(1) The PRQ message from gatekeeper to *MessageHandler*

H323	
SERVER_NAME:	d254057.cs.stir.ac.uk
caller:	6000
callee:	7000
user:	7000
destCallSignalAddress:	139.153.254.57:1721
srcCallSignalAddress:	139.153.254.62:1733
conferenceID:	27A24B2B 58EF1810 87810003 472E970D
conferenceGoal:	create
callType:	pointToPoint
callIdentifier:	Sequence

endpointIdentifier:	7067_endp
mediaWaitForConnect:	FALSE
canOverlapSend:	FALSE
multipleCalls:	FALSE

The *PRQ* message is a structured string with variable-value pairs. Each pair is listed on an independent line. All values are simple strings.

(2) The message from *MessageHandler* to *ProtoMsgHandler*

Comparing Figure 2 and Figure 3, we can see the message from *MessageHandler* to *ProtoMsgHandler* is obtained through removing the protocol header (H323) from the original *PRQ* message. The policy server chooses the *H323MsgHandler* according to the protocol type.

SERVER_NAME:	d254057.cs.stir.ac.uk
caller:	6000
callee:	7000
user:	7000
destCallSignalAddress:	139.153.254.57:1721
srcCallSignalAddress:	139.153.254.62:1733
conferenceID:	27A24B2B 58EF1810 87810003 472E970D
conferenceGoal:	create
callType:	pointToPoint
callIdentifier:	Sequence
endpointIdentifier:	7067_endp
mediaWaitForConnect:	FALSE
canOverlapSend:	FALSE
multipleCalls:	FALSE

(3) The message from *ProtoMsgHandler* to *H323MsgHandler*

This message is the same as for (2).

(4) The message from *H323MsgHandler* to *PolicyApplicator*

SERVER_NAME:	d254057.cs.stir.ac.uk	-- String
caller:	6000	-- LinkedList
callee:	7000	-- LinkedList
user:	7000	-- String
destCallSignalAddress:	139.153.254.57:1721	-- LinkedList
srcCallSignalAddress:	139.153.254.62:1733	-- LinkedList
conferenceID:	27A24B2B 58EF1810 87810003 472E970D	-- LinkedList
conferenceGoal:	create	-- LinkedList
conferenceType:	pointToPoint	-- LinkedList
callIdentifier:	Sequence	-- LinkedList
endpointIdentifier:	7067_endp	-- LinkedList
mediaWaitForConnect:	FALSE	-- LinkedList
canOverlapSend:	FALSE	-- LinkedList
multipleCalls:	FALSE	-- LinkedList

H323MsgHandler modifies two aspects of the message. One is to map the terminology from H.323 to policy. For example, *callType* is changed to *conferenceType*. The other is to set up some variable-value pairs into a Java structured type *LinkedList*.

(5) The response message from *PolicyApplicator* to *H323MsgHandler*

forward_to(1000)	-- LinkedList

The response message is set up in a *LinkedList*. Moreover, the terminology is that of policies.

(6) The response message from *H323MsgHandler* to *ProtoMsgHandler*

forward(1000) -- String

H323MsgHandler implements the inverse procedure for the response message. For example, *forward_to* (policy terminology) is changed to *forward* (H.323 terminology).

(7) The response message from *ProtoMsgHandler* to *MessageHandler*

This response message is the same as for (6).

(8) The *PRS* message from *MessageHandler* to gatekeeper

The *PRS* message is the same as for (6).

Abbreviations

APDU	Application Protocol Data Unit
ARQ/ACF/ARJ	Admission Request/Confirm/Reject
BRQ/BCF/BRJ	Bandwidth Request/Confirm/Reject
DRQ/DCF/DRJ	Disengage Request/Confirm/Reject
GRQ/GCF/GRJ	Gatekeeper Request/Confirm/Reject
IRQ/IRR	InfoRequestResponse
LRQ/LCF/LRJ	Location Request/Confirm/Reject
MCU	Multi-Control Unit
PRQ/PRS	Policy Request/Response
RAS	Registration, Admission and Status
RRQ/RCF/RRJ	Registration Request/Confirm/Reject

References

[1] ITU-T. *Packet-Based Multimedia Communications Systems*, Recommendation H.323, International Telecommunications Union, Geneva, Nov. 2000

[2] ITU-T. *Call Signalling Protocols and Media Stream Packetization For Packet-Based Multimedia Communication Systems*, Recommendation H.225, International Telecommunications Union, Geneva, Nov. 2000

[3] S. Reiff-Marganiec and K. J. Turner. Use of logic to describe enhanced communications services, FORTE 2002, pages 130–145, Nov. 2002

[4] S. Reiff-Marganiec and K. J. Turner. A policy architecture of enhancing and controlling features, Feature Interactions in Telecommunications and Software Systems VII, pages 239–246, Jun. 2003.

[5] GNU Gk. http://www.gnugk.org, Apr. 2004.

[6] ITU-T. *Call Diversion Supplementary Service for H.323*, Recommendation H.450.3, International Telecommunications Union, Geneva, May. 2000

[7] ITU-T. *Call Hold Supplementary Service for H.323*, Recommendation H.450.4, International Telecommunications Union, Geneva, Sep. 1998

[8] ITU-T. *Call Intrusion Supplementary Service for H.323*, Recommendation H.450.11, International Telecommunications Union, Geneva, Nov. 2000

[9] S. Reiff-Marganiec. The ACCENT Policy Server, Technical Report CSM-164, Computing Science and Mathematics, University of Stirling, Nov. 2003.

[10] ITU-T. *Information Technology – Abstract Syntax Notation One (ASN.1): Specification of Basic Notation*, Recommendation X.680, International Telecommunications Union, Geneva, 1994.

[11] ITU-T. *Information Technology – ASN.1 Encoding Rules – Specification of Packed Encoding Rules (PER)*, Recommendation X.691, International Telecommunications Union, Geneva, 1994.

[12] S. Reiff-Marganiec. ACCENT Project Policy Environment/Language, Technical Report CSM-161, Computing Science and Mathematics, University of Stirling, Nov. 2003.

Appendix: H.323–SIP Interworking

This appendix provides brief development notes on the supplementary work undertaken to link SIP devices into an H.323 network.

1. Development environment and tools

Operating system:	Redhat Linux 9.0
Edit software:	Source-Navigator 5.1.2
Debugging tool:	ddd v3.3, GNU gdb 20030609
Compile tool:	GNU Make version 3.79.1

2. Underlying support library

(1) pwLib: Portable Window Library, version 1.4.11

(2) openh323: the underlying infrastructure for H.323 protocol, version 1.11.7

3. Related websites and documents

(1) www.openh323.org

(2) www.gnugk.org

(3) www.vovida.org

(4) L. Dang, C. Jennings and D. Kelly. *Practical VoIP Using VOCAL*, O'Reilly.

(5) www.gnugk.org/h323manual.html (manual for *openh323gk*)

4. Infrastructure software

(1) openh323gk

(2) siph323csgw

5. Three aspects of the H.323 policy work

(1) A SIP-attached H.323 network

A SIP-attached H.323 network refers to an H.323 network that has no H.323 management device gatekeeper and uses a SIP server. All H.323 terminals register with the SIP registration server through *siph323csgw*. The key issue is how to assign an alias for an H.323 terminal. Normally, an H.323 terminal does not have a SIP-like address resembling an email address, but a simple alias or a number. So, *siph323csgw* adds its own IP address or domain name to the end of the H.323 terminal's alias or number. Suppose the terminal registers with the SIP proxy server as 2000@139.153.254.57.

In this architecture, the SIP Server is responsible for enforcing both SIP policies and H.323 policies. The *siph323csgw* package is the main focus. Because the original version of *siph323csgw* had some problems, these bugs had to be removed.

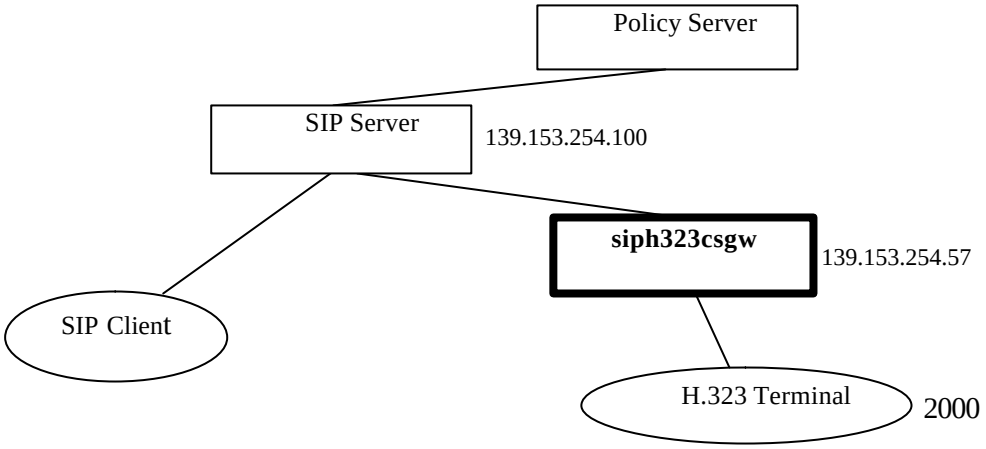

Figure 32. Policy Architecture for A SIP -Attached H.323 Network

(2) An independent H.323 network

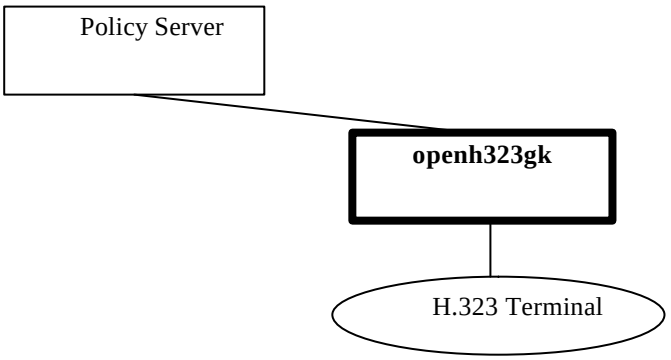

Figure 33. An Independent H.323 Network

The openh323gk is a gatekeeper managing the H.323 Zone. It is the key of this research project. All H.323 terminals will register with the openh323gk. The openh323gk monitors all H.323 calls, communicates with the policy server and enforce the H.323 policies.

(3) The enhanced *siph323csgw* between SIP network and H.323 network

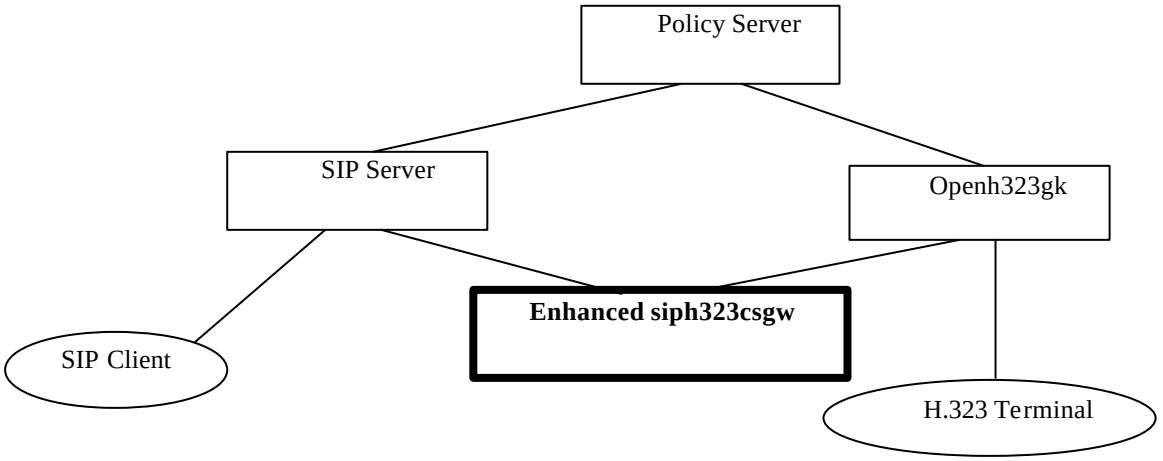

Figure 34. Policy architecture for A SIP-Attached H.323 Network

The SIP network and the H.323 network are relatively independent including their management servers. But when communication between them happens, a gateway responsible for translating is required. The current version of siph323csgw has not such a function. We need to enhance it.

6. Developments by the author

(1) Added the policy-supported part in *openh323gk*

(2) Added the H.323-supported part in the policy server including the code, the mapping database, and some H.323 policies written in APPEL

(3) Fixed the bugs of *siph323csgw* in order to construct the policy architecture of the SIP-attached H.323 network