A P2P AUDIO CONFERENCING APPLICATION USING H.323 VoIP

CALIFORNIA SOFTWARE LABS
REALIZE YOUR IDEAS

California Software Labs
6800 Koll Center Parkway,
Suite 100 Pleasanton
CA 94566, USA.
Phone  (925) 249 3000
Fax     (925) 426 2556
info@cswl.com
http://www.cswl.com
A P2P AUDIO CONFERENCING APPLICATION USING H.323 VoIP

A Technical Report

Technical Expertise Level: Intermediate

INDEX

INTRODUCTION ............................................................................................................. 5
VOICE OVER INTERNET PROTOCOL (VOIP) .......................................................... 5
WHAT IS H.323? ............................................................................................................. 5
FEATURES OF H.323 ...................................................................................................... 6
PORTABLE WINDOWS LIBRARY ............................................................................. 6
PROTOCOLS INCLUDED IN THE H.323 SPECIFICATION ...................................... 7
PROTOCOLS CONTROLLING CALL SETUP AND MANAGEMENT: ............................... 7
H.225 Call Signaling .................................................................................................... 7
Q.931 ........................................................................................................................... 8
H.245 Control Signaling................................................................................................. 8
PROTOCOLS USED IN REAL-TIME DATA TRANSFER .............................................. 9
T.120 Data Communications ....................................................................................... 9
RAS (Registration/Admission/Status) ......................................................................... 9
RTP (Real Time Protocol) ............................................................................................ 9
RTCP .......................................................................................................................... 10
AUDIO ENCODING ................................................................................................... 10
VIDEO ENCODING ................................................................................................... 10
COMPONENTS DEPLOYED IN H.323 ................................................................... 11
H.323 TERMINAL .................................................................................................... 11
GATEWAY ................................................................................................................. 12
GATEKEEPER ............................................................................................................ 13
MANDATORY GATEKEEPER FUNCTIONS .............................................................. 13
Address Translation ................................................................................................... 13
ADMISSION CONTROL ............................................................................................. 14
Bandwidth Control .................................................................................................... 14
Zone Management ..................................................................................................... 14
MULTIPOINT CONTROLLER ................................................................................... 14
MULTIPOINT PROCESSOR ....................................................................................... 14
MULTIPOINT CONTROL UNIT ................................................................................ 15
Centralized multipoint conferences ........................................................................ 15

CSWL Inc, Pleasanton, California
This paper targets on providing a basic understanding of Point-to-Point Audio conferencing using VoIP. This paper also presents a case study of an application that uses H.323 protocol stack from openH323.org, that aims to customize the protocol by giving the stack, and PWLIB, a portable Windows library, for performing Point to Point Audio Conferencing using VoIP. H.323 is an ITU (International Telecommunications Union) protocol suite that supports audio and
video transmission over the Internet. VoIP is part of this suite. PWLIB provides a set of portable Classes for use with Windows and Linux platforms.

It also presents source code examples of a Simple P2P audio conferencing application in both Windows and Linux platforms.

The Windows Application is developed in VC++ using MFC (Dialog Based) for audio conferencing.
The Linux Application is a simple command line application developed in C++.

The Source code for Linux is the same as Windows application except that it didn’t use MFC and runs as a Console application.
Introduction

VoIP is a term used for a set of facilities for managing the delivery of voice information using the Internet Protocol. This means sending voice information in digital form in discrete packets rather than using the conventional connection oriented circuit-switched networks. With VoIP, there is no dedicated connection between the communicating devices. In addition to IP, VoIP uses the real-time protocol (RTP) to help ensure that packets get delivered in a timely way.

Voice over Internet Protocol (VoIP)

IP cannot carry analog signals. VoIP works by converting the analog voice signal into a compressed digital data stream. This digital data stream is then fragmented into packets for transmission over the IP network to the destination where the packets are converted back into analog signal for reception.

Voice (source)- -> ADC - - - -> Internet - - -> DAC - -> Voice (dest).

What is H.323?

H.323 is an ITU (International Telecommunications Union) recommended standard, which provides a foundation for audio, video and data communications on non-guaranteed Quality of Service networks, or precisely in Local Area Networks (LAN) and IP based networks, including the Internet. H.323 is not a single protocol. It is a recommended specification for a range of protocols, which performs all the functions necessary to establish and maintain real time audio/video/data conferencing sessions over IP data networks.
Features of H.323

- Interoperability
- Multipoint support
- Network and Platform Independent
- Bandwidth Management
- Multicast Support
- Flexibility

Portable Windows Library

Portable Windows Library (PWLIB) is a class library that is used to produce applications to run on both Microsoft Windows and Unix X-Windows systems. It consists of several parts:

- A number of generic classes like container classes, a string class etc that are contained inside cglib (Container and Generic class Library). These are compiled from the same sources for every platform and there are no differences in implementation across the platforms.
- A number of OS specific classes that are platform specific like threads (PThread), processes (PProcess) and semaphores (PSemaphore) which are contained inside ptlib (Portable Threads, processes and inter-process communication Library) were put here because pwlib is meant to be a portable library, so the OS specifics need to be abstracted in a portable way.
- A number of network and multimedia classes like PSocket, PChannel, PsoundChannel, which are contained inside pnlib (Portable Network Library) and pmlib (Portable Multimedia Library) were also put here so they could be abstracted in a portable way.
- The ASN parser, which is not part of the library but a separate application, which is needed for compiling OpenH323 Library.
Protocols included in the H.323 specification

H.323 is a protocol suite that supports a range of protocols that provides component descriptions, signaling procedures, call control, system control, audio/video codecs, data protocols etc.

The figure above shows the range of protocols supported by the H.323 standard. The H.323 protocol suite includes a core "Conference Manager" layer to manage all conference setup activities. The "Conference Manager" is comprised of:

- An H.225 layer that converts streams to packets and synchronizes them during a session.
- An H.245 layer that is used to control communications between the terminal equipment. It specifies how messages are exchanged and how the user should interact with the teleconferencing application.

Protocols controlling call setup and management:

**H.225 Call Signaling**

The H.255 standard defines a layer that formats the transmitted video, audio, data, and control streams for output to the network, and retrieves the corresponding streams when they are needed.
streams from the network. As part of audio and video transmissions, H.225 uses the packet format specified RTP and RTCP specifications for the following tasks:

- Logical framing - defines how the protocol frames the audio and video data into packets for transport over a selected communications channel.
- Sequence numbering - determines the order of data packets transported over a communications channel.
- Error detection.

**Q.931**

This protocol defines how each H.323 layer interacts with peer layers, so that participants can interoperate with agreed upon formats. The Q.931 protocol resides within H.225. As part of H.323 call control, Q.931 is a link layer protocol for establishing connections and framing data. Q.931 provides a method for defining logical channels inside of a larger channel. Q.931 messages contain a protocol discriminator that identifies each unique message with a call reference value and a message type. The H.225.0 layer then specifies how these Q.931 messages are received and processed.

**H.245 Control Signaling**

This standard provides the call control mechanism that allows H.323-compatible terminals to connect to each other. The H.245 Control Channel is a reliable channel that carries control messages governing operation of the H.323 endpoint. These control messages carry information related to the following:

- Capabilities exchange
- Opening and closing of logical channels used to carry media streams
- Preference requests
- Flow-control messages
General commands and indications

There is only one H.245 Control Channel per call.

**Protocols used in Real-time data transfer**

*T.120 Data Communications*

T.120 is used when data conferencing is needed. T.120 makes it possible to share data and applications while teleconferencing. This T.120 support means that data handling can occur either in conjunction with H.323 audio and video, or separately.

*RAS (Registration/Admission/Status)*

The H.225.0 standard also includes registration, admission, and status (RAS) control. RAS is the protocol between endpoints (terminals and gateways) and gatekeepers, which makes the connections between them available. The RAS is used to perform registration, admission control, bandwidth changes, status, and disengage procedures between endpoints and gatekeepers. RAS is not used if a Gatekeeper is not present.

*RTP (Real Time Protocol)*

RTP is a real-time transport protocol that provides end-to-end delivery services to support applications transmitting real-time data, e.g., interactive audio and video, over unicast and multicast network services. RTP services include payload type identification, sequence numbering, and time stamping. Delivery is monitored by means of a closely integrated control protocol called RTCP.

RTP is a real-time transport protocol that provides end-to-end delivery services to support applications transmitting real-time data, e.g., interactive audio and video, over unicast and multicast network services. RTP services include payload type
identification, sequence numbering, and time stamping. Delivery is monitored by means of a closely integrated control protocol called RTCP.

**RTCP**

The Real-Time Control Protocol (RTCP) is used for the control of RTP. RTCP monitors the quality of service, conveys information about the session participants, and periodically distributes control packets containing quality information to all session participants through the same distribution mechanisms as the data packets.

**Audio encoding**

H.323 supports proven ITU standard audio codec algorithms, including G.711 for speech, which transmits voice at 56 or 64 Kbps. Support for other ITU voice standards (G.722, G.723, G.728, G.729) is optional, since each one reflects tradeoffs between speech quality, bit rate, computing power, and signal delay.

The following codecs are supported in the H.323 specification.

- G.711: Audio Codec, 3.1 KHz at 48, 56, and 64 Kbps (normal telephony)
- G.722: Audio Codec, 7 KHz at 48, 56, and 64 Kbps
- G.728: Audio Codec, 3.1 KHz at 16 Kbps
- G.723: Audio Codec, for 5.3 and 6.3 Kbps modes
- G.729: Audio Codec.

**Video encoding**

- H.261: Video Codec for audiovisual services at P x 64 Kbps
- H.263: Specifies a new video codec for video POTS.
However, when a particular application calls for compression and decompression of audio and video streams, the specified codecs must also be included within the terminal equipment.

Components Deployed in H.323

Components of an H.323 solution may include but are not limited to terminals, gateways, gatekeepers, multipoint controllers (MCs), multipoint processors (MPs), and multipoint control units (MCUs). This section discusses the role of each element in an overall H.323 system as part of an end-to-end network.

H.323 Terminal

A H.323 terminal is nothing but an endpoint, which can communicate with other endpoint, Gateway, Gatekeeper or MCU.

All H.323 terminals must support these features:

- H.245, a standard for negotiating channel usage and capabilities
- Q.931, a standard for call signaling and setup
- Registration/Admission/Status (RAS), a protocol for communicating with gatekeepers
- RTP/RTCP support, for sequencing audio and video packets.

H.323 terminals may optionally support these features:

- Video codecs
- T.120 data conferencing protocols
- MCU capabilities
- Gateways
Gateway

A gateway connects two dissimilar networks. An H.323 gateway provides connectivity between an H.323 network and a non-H.323 network. For example, a gateway can connect and provide communication between an H.323 terminal and SCN networks (SCN networks include all switched telephony networks, e.g., public switched telephone network [PSTN]). This connectivity of dissimilar networks is achieved by translating protocols for call setup and release; converting media formats between different networks, and transferring information between the networks connected by the gateway. A gateway is not required, however, for communication between two terminals on an H.323 network.

In general, the purpose of the gateway is to reflect the characteristics of a LAN endpoint to an SCN endpoint and vice versa. On the H.323 side, a gateway runs H.245 control signaling for exchanging capabilities, H.225 call signaling for call setup and release, and H.225 registration, admissions, and status (RAS) for registration with the gatekeeper. On the SCN side, a gateway runs SCN-specific protocols (e.g., ISDN and SS7 protocols). Terminals communicate with gateways using the H.245 control-signaling protocol and H.225 call-signaling protocol. The gateway translates these protocols in a transparent fashion to the respective counterparts on the non-H.323 network and vice versa. The gateway also performs call setup and clearing on both the H.323 network side and the non-H.323 network side. Translation between audio, video, and data formats may also be performed by the gateway. Audio and video translation may not be required if both terminal types find a common communications mode.

For example, in the case of a gateway to H.320 terminals on the ISDN, both terminal types require G.711 audio and H.261 video, so a common mode always exists. The gateway has the characteristics of both an H.323 terminal on the H.323 network and the other terminal on the non-H.323 network it connects. A gateway may be able to support several simultaneous calls between the H.323 and non-
H.323 networks. In addition, a gateway may connect an H.323 network to a non-H.323 network. A gateway is a logical component of H.323 and can be implemented as part of a gatekeeper or a MCU.

Gatekeeper

The gatekeeper is the most important of the H.323 components. The gatekeeper's primary job is to act as the central point for all calls within its zone and provide call control services for registered H.323 endpoints.

Gatekeepers in H.323 networks are optional, but if they are present in the network endpoints must use their services. Although they are not required, gatekeepers provide important services such as addressing, authorization and authentication of terminals and gateways; bandwidth management; accounting; billing; and charging and may also provide call-routing services.

The H.323 standards define mandatory services that the gatekeeper must provide and specifies other optional functionality that it can provide.

Mandatory Gatekeeper Functions

Address Translation

Calls originating within an H.323 network may use an alias to address the destination terminal. Calls originating outside the H.323 network and received by a gateway may use an E.164 telephone number to address the destination terminal. The gatekeeper must be able to translate the alias or the E.164 telephone number into the network address for the destination terminal. The destination endpoint can be reached using the network address on the H.323 network. The translation is done using a translation table that is updated with Registration messages.
Admission Control

The gatekeeper can control the admission of the endpoints into the H.323 network. It uses RAS messages, admission request (ARQ), confirm (ACF), and reject (ARJ) to achieve this. Admissions control may also be a null function that admits all requests.

Bandwidth Control

Gatekeepers must support the RAS bandwidth messages. How they provide the bandwidth access or bandwidth management, however, is left to the service provider or enterprise manager's individual policy. For instance, if a network manager has specified a threshold for the number of simultaneous connections on the H.323 network, the gatekeeper can refuse to make any more connections once the threshold is reached. The result is to limit the total allocated bandwidth to some fraction of the total available, leaving the remaining bandwidth for data applications. In many cases, any bandwidth requests will be honored, unless the network or particular gateway is congested.

Zone Management

A gatekeeper is required to provide the above functions-address translation, admissions control, and bandwidth control-for terminals, gateways, and MCU located within its zone of control.

Multipoint Controller

A multipoint controller (MC) is a LAN-based H.323 entity that controls three or more terminals participating in multipoint conferences. The MC negotiates between terminals in order to know what audio or video coder/decoder (CODEC) to use, and also manages conference resources, by determining which, if any, of the audio and video streams will be multicast.
Multipoint Processor

A multipoint processor (MP) is a LAN-based H.323 entity that centrally processes audio, video, and/or data streams in a multipoint conference.

Multipoint Control Unit

MCU provide support for conferences of three or more H.323 terminals (endpoints). All terminals participating in the conference establish a connection with the MCU. Under H.323, an MCU consists of a Multipoint Controller (MC) and zero or more Multipoint Processors (MPs).

The H.323 recommendation uses three concepts of multipoint conferences: Centralized and decentralized multipoint conferences, and Hybrid multipoint conferences, which use a combination of centralized and decentralized features.

Centralized multipoint conferences

This concept requires an MCU to facilitate a multipoint conference. All terminals send audio, video, data and control streams to the MCU in a point-to-point fashion. The MC controls the conference using H.245 control functions. The MP mixes the audio, distributes data, switches and mixes video and sends the resulting streams back to the participating terminals. The MP may also converse between different codecs and bit rates and may use multicast to distribute processed video.

Decentralized multipoint conferences

This concept makes use of multicast technology. In this concept, terminals that participate in a conference, multicast audio and video to other terminal that also participate in the conference, without sending the data to an MCU, but, still, the control is processed by the MCU, and H.245 Control Channel information is still
transmitted in a point-to-point mode to an MC. Receiving terminals are responsible for processing the incoming audio and video streams, and they indicate the MC how many simultaneous video and audio streams they can decode. The MP can provide video selection and audio mixing in a decentralized multipoint conference.

**Hybrid multipoint conferences**

As described, this concept uses a combination of centralized and decentralized features. H.245 signals and an audio or video stream are processed through point-to-point messages to the MCU. The remaining signal (audio or video) is transmitted to participating H.323 terminals through multicast. H.323 also supports mixed multipoint conferences in which some terminals are in a centralized conference, others are in a decentralized conference, and an MCU provides the bridge between the two types.
A connection is initiated from the caller through the H.323 port, usually port 1720, by sending a connection request packet to its peer. The type of transport is chosen by H.323. The H.245 in remote machine decides which audio and video codecs to use, and makes decisions on the commands that cause the connection to be made. The H.245 session then executes an OpenLogicalChannel sequence. The sequence is a series of initialization routines that are called one after another in a particular order to open the channels for communication with the peer.

A half-duplex channel is opened for RTP, the actual data stream, and a full-duplex channel for RTCP, which has control information. The associated RTCP and RTP streams are required to be one port apart with the RTP port being even and the RTCP being the next higher odd.

The key component is the Jitter buffer (JB). The packet network can deliver packets out of order, duplicated, corrupted, or not at all. Packet delivery also tends to be bursty. The JB reassembles the incoming packets into a continuous stream such that the decompression algorithms can convert the coded data back into linear data.

JBs fall into two main categories:
- Static
- Dynamic.

Static JB has a fixed length window, which introduces a particular delay into the system. This delay allows late (out of order) packets to arrive and be slotted into position. If the packet network delay increases beyond the window, performance falls dramatically.

A dynamic JB monitors various parameters, such as average packets loss, and adjusts the window size during the call. This has to be done carefully, because changes in delay are very noticeable to the user. However, a dynamic solution is generally preferable to the alternatives, which is either a large fixed window with a large delay, or a small fixed window with corresponding high levels of packet loss.

OpenH323 Class Library used in Audio Conferencing Application

This is a Open Source class library that has been used for the development of our P2P Audio Conferencing Application over packet based networks.
- **PObject**: Ultimate parent class for all objects in the class library.
- **PChannel**: Abstract class defining I/O channel semantics.
- **Process**: Derived from PThread, represents an Operating System Process.
- **PThread**: Derived class of PObject, which defines a thread of execution in the system.
- **PIndirectChannel**: This is a channel that operates indirectly through another channel(s).
- **H323EndPoint**: This class manages the H323 endpoint
- **H323Connection**: This class represents a particular H323 connection between two endpoints.
- **H323Channel**: This class describes a logical channel between the two endpoints.
- H323Capability: This class describes the interface to a capability of the endpoint, usually a codec, used to transfer data via the logical channels opened and managed by the H323 control channel.
- H323Codec: This class embodies the implementation of per specific codec instance.
- H323Listener: This class describes a “listener” on a transport protocol.
- RTPSession: This class is for encapsulating the IETF Real Time Protocol interface.
- H323GateKeeper: This class embodies the H2250 RAS protocol to gatekeepers.

Basic Requirements of our P2P Audio Conferencing Application

Using your PC and the Internet, you can now hold conversations with friends and family, and collaborate with co-workers around the world. For our Point-to-Point Audio Conferencing Application, we need at least two PC’s with

- Sound Cards, Microphone and Speakers
- OpenH323 stack downloadable from OpenH323.org
- Linux and Windows Operating Systems
- Audio Codecs

We have given options for the following codecs:

1. Microsoft GSM 6.10 Audio Codec
2. Microsoft CCITT G.711 A-law and u-law Audio Codec – PCM audio codec 56/64 Kbps
3. Microsoft G.723.1 Audio Codec
4. LPC-10 Codec
The application makes use of the codec that has the highest priority among the different codecs available on the PC.

**Sequence Of Steps Involved In Our P2P Audio Conferencing Application**

**Endpoint Initialization**

A set of routines initializes the endpoint by specifying the codecs that the application is capable of and some basic initializations like the kind of play device and record device used.

```cpp
BOOL CMfcEndPoint::Initilise (CMfcDlg *dlg)
{
    m_dialog = dlg;
    m_dialog->m_caller.SetWindowText("Waiting for incoming call.....");
}
```

CSWL Inc, Pleasanton, California - 21 -

A P2P AUDIO CONFERENCING APPLICATION USING H.323 VoIP
//Initialising the required parameters for audio channel
silenceDetect=FALSE;       // Enable/Disable silence detection.
silenceDeadband=3200;  // 400 milliseconds of silence needed
signalDeadband=480;    // 60 milliseconds of signal needed

/** Set the initial bandwidth for the channel. */
* This calculates the initial bandwidth required by the channel and
* returns TRUE if the connection can support this bandwidth.
* The default behaviour gets the bandwidth requirement from the codec
* object created by the channel.
*/

SetInitialBandwidth(10000);

//Setting the Sound device for the endpoint
SetSoundChannelRecordDevice(GetSoundChannelRecordDevice());
SetSoundChannelPlayDevice(GetSoundChannelPlayDevice());

/**Set the default sound channel buffer depth. */
SetSoundChannelBufferDepth(GetSoundChannelBufferDepth());

/**Set the default maximum audio delay jitter parameter. */
SetMaxAudioDelayJitter(120);

/**Set the IP Type Of Service byte for RTP channels. */
SetRtpIpTypeofService(GetRtpIpTypeofService());

//end of initializing
```c
#ifdef WIN32
    H323_ACMG7231Capability * acmG7231 = new H323_ACMG7231Capability();
    if (acmG7231->IsValid())
        SetCapability(0, 0, acmG7231);
    else
        delete acmG7231;
#endif

/** Add a codec to the capabilities table. This will assure that the
 * assignedCapabilityNumber field in the codec is unique for all codecs
 * installed on this endpoint.
 * If the specific instance of the capability is already in the table, it
 * is not added again. There can be multiple instances of the same
 * capability class however.
 */

AddCapability(new H323_GSM0610Capability);
AddCapability(new H323_G711Capability(H323_G711Capability::muLaw));
AddCapability(new H323_G711Capability(H323_G711Capability::ALaw));

/** The function sets the capability descriptor lists.
 If descriptorNum is P_MAX_INDEX (integer constant), the next available index in
 the
 array of descriptors is used. Similarly if simultaneous is P_MAX_INDEX then the
 nextavailable SimultaneousCapabilitySet is used.

 This function returns the PINDEX (integer constant).
 */
SetCapability(0, 0, new H323_LPC10Capability(*this));
SetCapability(0, 0, new H323_GSM0610Capability);
SetCapability(0, 0, new MicrosoftGSMAudioCapability);
```
SetCapability(0, 0, new H323_G711Capability(H323_G711Capability::ALaw));
SetCapability(0, 0, new H323_G711Capability(H323_G711Capability::muLaw));
H323_UserInputCapability::AddAllCapabilities(capabilities, 0, 0);

return StartListener(new H323ListenerTCP(*this));
}

After initiating, the endpoint is ready for accepting or making a call.

Initiate a Call

Endpoint 1 will initiate a call to Endpoint 2 by invoking the OnCall() function.

Void CMfcDlg::OnCall()
{
    //Variable to store the ip address
    PString fullAddress = (const char *)m_destination;

    //Checks for the gateway, if it is not empty, it will add
    //to its end of the address
    if (!m_gateway.IsEmpty())
    {
        fullAddress += '@' + m_gateway;
    }

    /** Makes call to the remote machine
    * internally calls the BuildConnectionToken function to get the token
    * that contains the connection information.
    * An appropriate transport is determined from the remoteParty parameter.
    */
The general form for this parameter is *[alias@][transport$]host[:port] where the
default alias is the same as *the* host, the default transport is "ip" and the
default port is 1720.

```cpp
m_endpoint.MakeCall((const char *)fullAddress, m_token);
```

Create a Connection

Endpoint1 calls this function and the reference ID of this is passed to Endpoint2.

```cpp
H323Connection * CMfcEndPoint::CreateConnection(unsigned int refID)
{
    char cArray[20];  //Character array to store the refID
    sprintf(cArray,"Ref id is-> %d",refID);
    m_dialog->m_caller.SetWindowText("Connection Created...");

    return new H323Connection(*this, refID);
}
```

Connection Establishment

After receiving the reference ID from Endpoint1 the following sequence of steps
occur at Endpoint2:

- The function OnIncomingCall is invoked. It gives an opportunity for an
  application to alter the reply before transmission to the other endpoint. If
FALSE is returned the connection is aborted and a Release Complete PDU is sent.

```cpp
BOOL CMfcEndPoint::OnIncomingCall
(
    H323Connection & connection, // Connection that was established
    const H323SignalPDU & setupPDU, // Received setup PDU
    H323SignalPDU & alertingPDU   // Alerting PDU to send
) {

    PString PStrFileName; //Variable for storing the filename
    CString slash="\";
    PStrFileName=CString("c:\") + CString(slash) + CString("winnt")
    + CString(slash) +CString("media") + CString(slash)+
    CString("ringin.wav");
    //PSound for playing the sound (i.e. *.wav files)
    PSound::PlayFile(PStrFileName, FALSE);
    return TRUE;
}
```

- AnswerCallResponse is a callback for answering an incoming call. If AnswerCallDenied is returned the connection is aborted and a Release Complete PDU is sent. If AnswerCallNow is returned then the H.323 protocol proceeds. Finally if AnswerCallPending is returned then the protocol negotiations are paused until the AnsweringCall() function is called. The default behaviour simply returns AnswerNow.

```cpp
H323Connection::AnswerCallResponse
CMfcEndPoint::OnAnswerCall(H323Connection & connection, const
PString
&caller, const H323SignalPDU &, H323SignalPDU &)
```
m_dialog->m_token = (const char *) connection.GetCallToken();
m_dialog->m_caller.SetWindowText(caller + " is calling.");

return H323Connection::AnswerCallNow;
}

• Open a channel for use by an audio codec. The H323AudioCodec class will use this function to open the channel to read/write PCM data.

BOOL CMfcEndPoint::OpenAudioChannel(H323Connection & /*connection*/,
BOOL isEncoding, unsigned bufferSize, H323AudioCodec & codec)
{
AfxMessageBox("Entered OpenAudioChannel of Endpoint");
/* Variable to store the device name */

PString deviceName;
if (isEncoding)
{
    deviceName = (const char *) GetSoundChannelRecordDevice();
}
else
    deviceName = (const char *) GetSoundChannelPlayDevice();

PSoundChannel * soundChannel = new PSoundChannel;
if (soundChannel->Open(deviceName, isEncoding ? PSoundChannel::Recorder
: PSoundChannel::Player, 1, 8000, 16))
{
    AfxMessageBox(" device opened");
    soundChannel->SetBuffers(bufferSize, soundChannelBuffers);
return codec.AttachChannel(soundChannel);

PTRACE(1, "Codec\tCould not open sound channel " << deviceName
    << ",\" for " << (isEncoding ? "record" : "play")
    << "ing: " << soundChannel->GetErrorText());
delete soundChannel;
return FALSE;
}

• The function OnStartLogicalChannel is a Call back for opening a logical channel. First a logical channel is opened for the transmitter (Endpoint1) and then for the receiver (Endpoint2).

BOOL CMfcEndPoint::OnStartLogicalChannel(H323Connection &
    connection,
    H323Channel & channel )
{
    myLogicalChannel(channel,1, 2);

    return true;
}

const H323SignalPDU & setupPDU, // Received setup PDU
    H323SignalPDU & alertingPDU     // Alerting PDU to send
    )
{
    PString PStrFileName; //Variable for storing the filename
    CString slash="\";
    PStrFileName=CString("c:" ) + CString(slash) + CString("winnt") +
        CString(slash) +CString("media") + CString(slash)+
        CString("ringin.wav");
    //PSound for playing the sound (i.e. *.wav files)
The function myLogicalChannel determines the codec of transmitter and receiver.

```cpp
void CMfcEndPoint::myLogicalChannel(const H323Channel & channel, unsigned txStrID, unsigned rxStrID)
{
    const H323Capability & capability = channel.GetCapability();

    /*To store the name of the codec*/
    PString name = capability.GetFormatName();

    /*To store the frames*/
    PString frames;

    if (capability.IsDescendant(H323AudioCapability::Class()))
    {
        unsigned numFrames = channel.GetDirection() == H323Channel::IsTransmitter
            ? ((H323AudioCapability&)capability).GetTxFramesInPacket()
            : ((H323AudioCapability&)capability).GetRxFramesInPacket();
        numFrames =
            ((H323AudioCapability&)capability).GetTxFramesInPacket();
        frames.printf(" (%u frames)", numFrames);
    }
    switch (channel.GetDirection())
    {
        case H323Channel::IsTransmitter:
            AfxMessageBox("Transmitter");
            AfxMessageBox((const char *)name);
            break;
    }
}
```

```cpp
PSound::PlayFile(PStrFileName, FALSE);
return TRUE;
}
```
void CMfcEndPoint::OnConnectionEstablished(H323Connection & connection, 
const PString & token)
{
    m_dialog->m_caller.SetWindowText("Connection Established");
    }

• The function OnConnectionEstablished is a call back function that is called whenever a connection is established. This indicates that a connection to a remote endpoint was established with a control channel and zero or more logical channels.
m_dialog->m_token =(const char *) token;
m_dialog->m_call.EnableWindow(FALSE);
m_dialog->m_answer.EnableWindow(FALSE);
m_dialog->m_refuse.EnableWindow(FALSE);
m_dialog->m_hangup.EnableWindow();
m_dialog->m_caller.SetWindowText("In call with "+
connection.GetRemotePartyName());
}

Data Transfer

The same sequence of steps in step 4 will occur in Endpoint1(caller) and the connection will be established. After the connection gets established the data transfer will take place, which is monitored by the function

```cpp
void CMfcEndPoint::OnRTPStatistics(const H323Connection & connection,
const RTP_Session & session)
const
{
    //For displaying the Packet information
    AfxMessageBox("Calling RTPStatistics");
    long lngGetPackReceived= session.GetPacketsReceived();
    long lngGetPackSent= session.GetPacketsSent();
    long lngGetPackLost= session.GetPacketsLost();
    char cArrayGetPackSent[200];
    sprintf(cArrayGetPackSent," Packets Sent : %ld \n Packets Received : %ld \n Packets Lost : ",lngGetPackSent,lngGetPackReceived,lngGetPackLost);
    m_dialog->m_packsentval.SetWindowText(cArrayGetPackSent);
    //For displaying the call duration
    PStringStream duration;//PString variable to store the duration of call
```
duration << setprecision(0) << setw(5) 
    << (PTime() - connection.GetConnectionStartTime());
  m_dialog->m_calldurationvalue.SetWindowText(duration);
}

The Endpoints can also pause the channel. A paused channel is one that prevents
the annunciation of the channels data. For example, for audio this would MUTE the
data, for video it would still frame. Note that channel is not stopped, and may
continue to actually receive data, it is just that nothing is done with it.

void CMfcDlg::OnMute()
{
  if(m_mute.GetCheck())
  {
    //Finds the current connection
    H323Connection * connection =
    m_endpoint.FindConnectionWithLock(m_token);
    if (connection == NULL)
    {
      AfxMessageBox("No conn found");
      return;
    }
    H323Channel * channel =
    connection->FindChannel(RTP_Session::DefaultAudioSessionID,TRUE);
    if (channel != NULL) {
      channel->SetPause(true);
    }
    connection->Unlock(); //unlocks the current connection
  }
  else
  {
    CSWL Inc, Pleasanton, California - 32 -
    A P2P AUDIO CONFERENCING APPLICATION USING H.323 VoIP


```
H323Connection * connection =
m_endpoint.FindConnectionWithLock(m_token);
if (connection == NULL)
{
    AfxMessageBox("No conn found");
    return;
}
char myst[50];  //To store the sessionID
sprintf(myst,"%u",RTP_Session::DefaultAudioSessionID);
AfxMessageBox(myst);
H323Channel * channel =
connection->FindChannel(RTP_Session::DefaultAudioSessionID,TRUE);
if (channel != NULL) {
    AfxMessageBox("Channel not empty");
    channel->SetPause(false);
}
connection->Unlock();   //unlocks the current connection
```

Text-based Chat

H.323 also facilitates sending texts between two endpoints. Endpoints make use of this function to transfer texts.

```
[Connection object].SendUserInput("PString value to be sent");
```

Call Hang-up

Any endpoint can hangup a call at any time. This is done by the function
Void CMfcDlg::OnHangup()
{
    m_endpoint.ClearCall(m_token);
    m_hangup.EnableWindow(FALSE);
    m_call.EnableWindow();
}

It clears a current connection. This hangs up the connection to a remote endpoint. 
Note that this function is asynchronous.

3-way conferencing using H.323

This scenario clearly explains how the 3-way conversation takes place using H.323 terminals and a Gatekeeper.

![](image)

**Explanation:**

- Terminal 1 will request a call to Terminal 2 through the Gatekeeper.
- GateKeeper checks up with the Registration /Admission/Status (RAS) and 
gives the IP address of Terminal 2 to Terminal 1
- Now after this Terminal 2 will use this IP address to directly converse to the 
  Terminal 2.
Call Connection:

Explanation:

- Terminal 1 creates a logical path between Terminal 1 and Terminal 2.
- Terminal 2 requests permission from gatekeeper to respond for the conversation.
- Gatekeeper responds for the call to Terminal 2.
- Terminal 2 issues an alert signal to Terminal 1 to connect. Terminal 1 establishes a connection with Terminal 2 through the IP address.

H.245 Connection:

Explanation:

- For this example, assume Terminal 1 acting as Master and become the Multipoint Controller (MC) of the conference.
Multipoint Communication:

**Explanation:**

- Terminal 1 invite Terminal 3 through the Gatekeeper.
- Gatekeeper resolves the IP address of Terminal 3 and sends it to Terminal 1.
- A call is setup between the two terminals 1 and 3.
- Terminal 3 request the Gatekeeper to join the conversation with Terminal 1.
- Gatekeeper then acknowledges Terminal 3’s request.
- Terminal 3 alert and connect to Terminal 1.
- H.245 connection is established between the Terminal 1 and Terminal 3.

**Logical Flow of Audio Packets**

Any MultiConferencing Application uses the same set of classes as the Point-to-Point Conferencing application.

At any point in time, there are N nodes connected to the MCU. Consequently, there are N copies of H323Connection class, which will be labelled connA, connB... connN. There is only ever one H323EndPoint. There are N*(N-1) instances of audio buffers. Each connection has a dictionary, containing (N-1) instances of audiobuffers. ConnI (Connection class of the Endpoint) has audio Buffers, labeled abA, abB, abC... (Not abI) ...abM, abN.
• **Incoming Audio (audio data arrives at the MCU)**
  - The audio codecs write to the Incoming Audio channel.
  - IncomingAudio sends data to connI.
  - ConnI writes the data to the Endpoint.
  - The Endpoint copies the data to connA, connB.. (Not connI)...connM, connN)
  - The connections listed in step d copy the data to the specified audio buffer.

Thus, audio data from connI is copied into abI for connA, copied into abI for connB, copied into abI for connC etc. Thus, audio data from connI is copied (N-1) times.

• **Outgoing Audio (the audio encoder requests audio data to send))**
  - The audio codec requests data from the Outgoing Audio channel.
  - The Outgoing Audio channel requests data from the connI.
  - ConnI requests data from the Endpoint.
  - The Endpoint’s ReadAudio method then finds the connection associated with audio codec that has requested data. - In this case connI.
  - The Connection’s ReadAudio method is then called for connI.
  - Connection’s ReadAudio combines the data in each of its audiobuffers, which is abA, abB, abC... (Not abI) ...abM, abN.

---

**Features of our P2P Audio Conferencing Application**

- GUI client for Windows
- Command line client for Linux
- GSM full rate (06.10), LPC-10 and G.711 Codecs
- Jitter buffering
- Silence Suppression
- User Indication Messages
- Mute
- RTP-Session Statistics

**Common Applications of H.323 protocol**

- Desktop Videoconferencings
- Internet Telephony and Video telephony
- Collaborative Computing
- Network Gaming
- Business Conference Calling
- Distance Learning
- Support and Help Desk Applications
- Interactive Shopping

**Issues Encountered in our Project**

**Windows Platform Issues**

- We got an error in PWLIB that unistd.h file was not found which has been included under include sections of the header files. We deleted that part of code where we encountered the inclusion of unistd.h file and it worked. We verified this with openh323.org where they confirmed that the file is necessary only for linux applications.

- We had problem in compiling H.323 library. In file codecs.h, rtp.h file has been included as a standard header file, which caused a lot of errors specifying that the classes and member functions could not be found in the file specified. Later we included that as a user defined header file, which removed those errors.

Replaced the line, #Include <rtp.h> to #include “rtp.h”
• We rectified an error in assigning values to PString variable in the PWLIB. Normally assigning a string to the PString variable causes the application to hang. Later, we found the source type of the PString class and typecast the assigned values to const char * which rectified some major problems in running the application such as getting the call established in the code

    PString m_token;
    OnAnswerCall( connection,caller,signalPDU)
    {
        m_token=(const char*) Connection.GetCallToken();
        //Other implementation Code
    }

Linux Platform Issues

• Compilation of OpenH.323 library in Red hat Linux version 7.0 could not proceed due to some incompatibilities while compiling the sound.h file. We reported the problem to openh323.org and were suggested that the compilation would proceed without any errors in Red hat Linux version 6.2. Accordingly, we proceeded with our Compilation using Red hat Linux version 6.2 and were able to successfully build the library.

Conclusion

It has been widely recognized that H.323 has the potential to become the dominant standard for the next generation of IP telephony and for teleconferencing over the Internet.

In this paper, we provided an overview on VoIP and H.323 family of protocols used for audio/video teleconferencing and how it may be utilized (or a portion of it) in the design of applications that enable teleconferencing over the Internet. The
patches of code under the logical sequence clearly explain how peer-to-peer audio transmission takes place between two terminals. It also throws light on the different functionalities provided by H.323 protocol stack. It can be further enhanced to provide multipoint audio/video-conferencing between two or more terminals.

**Terminology**

**Codec (Coder/Decoder):** Equipment to convert between analog and digital information format. Also may provide digital compression and switching functions. Codecs may be either a hardware or a software in H.323 we mainly use software codecs.

**ITU:** International Telecommunication Union. An international standards body, which is a committee of the ITU.

**RTP:** Real-Time Transport Protocol. It is the standard protocol for streaming applications developed within the Internet Engineering Task Force (IETF). In RTP both Video and audio are sent as two different streams to the client, which are identified by the Packet ID.

**RTCP:** The transport protocol RTP is augmented by RTCP, the real-time control protocol. Its primary function is to provide feedback on the quality of the data distribution. It includes timing reconstruction, loss detection, security and content identification.

**RAS:** Registration/Admission/Status.
References

H.323 protocol stack can be downloaded from the link below
http://www.openh323.org

Follow the links to get documentation on other technologies surrounding H.323 protocol
http://www.pulsewan.com/data101/h323_basics.htm
http://www.packetizer.com/iptel/h323
http://ils.unc.edu/~davil/inls191/Guru

For details about Jitter Buffer,
http://www.brunel.ac.uk/~ee97ssp2/jitter_buffer.htm

For typical H.323 call setup
http://www.protocols.com/voip/typical_h323_call.htm

For documentation of all classes in H.323
http://www.au.openh323.org/docs/OpenH323/ClassReference.html

For documentation on PWLIB class library,
http://www.openh323.org/docs/PWLib

---

Copyright Notice:

2002 California Software Labs. All rights Reserved. The contents on the document are not to be reproduced or duplicated in any form or kind, either in part or full, without the written permission of California Software labs. Product and company names mentioned here in are the trademarks of their respective companies.