Starting with VoIP

Mindfire Solutions

www.mindfiresolutions.com

March 5, 2002

Abstract:

This paper discusses VoIP from developer's perspective and looks into the core issues like packetization, recording and playback of voice data in real time. Many other issues like implementation of H.323 or SIP protocols are not discussed here. Neither the communication part involving RTP and RSVP.

Introduction

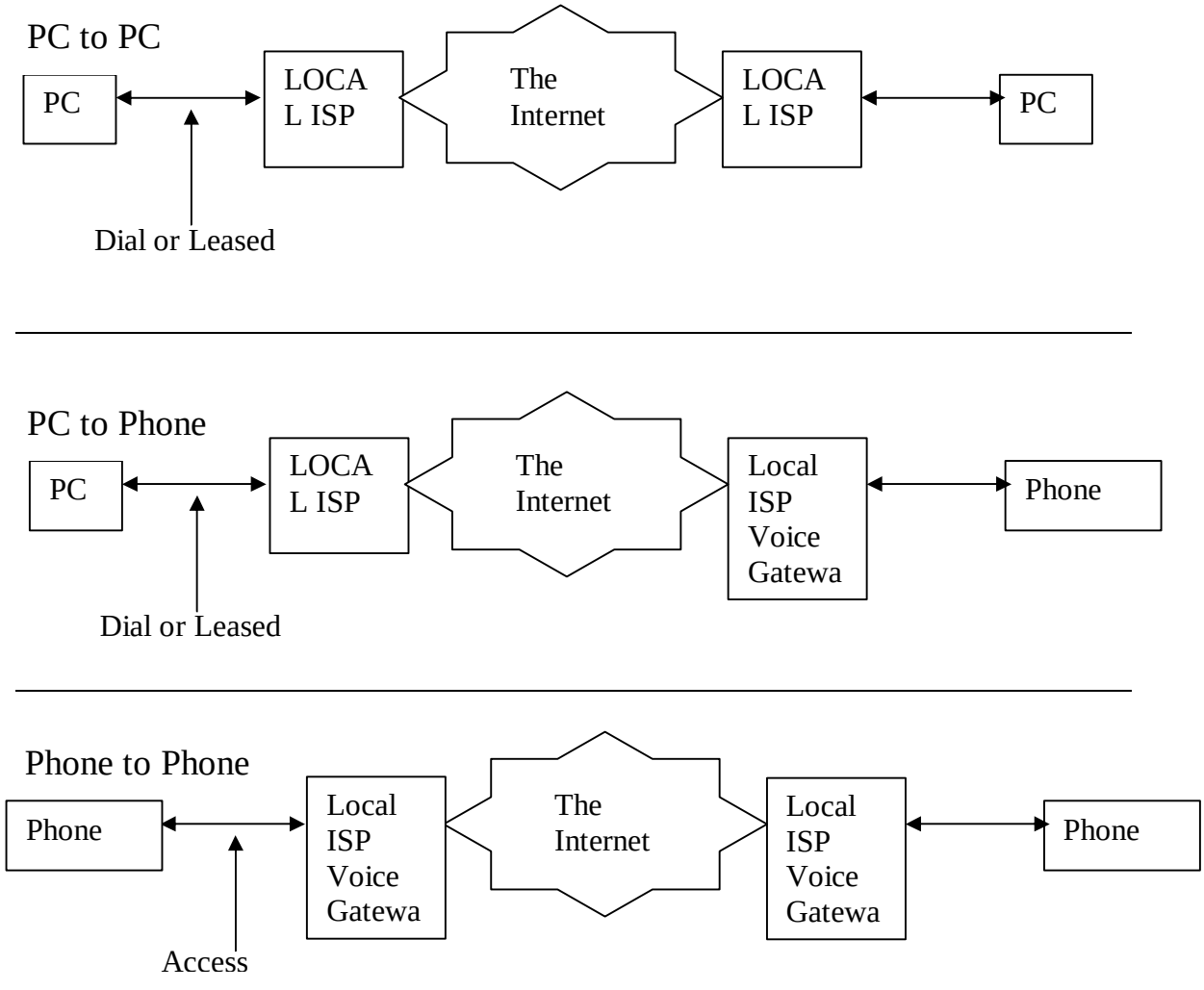
VoIP stands for 'V'oice 'o'ver 'I'nternet 'P'rotocol. As the term says VoIP tries to let go voice (mainly human) through IP packets and, in definitive through Internet. It is the collection of technologies that emulates and extends today's circuit-switched telecommunication services to operate on packet-switched data networks based on the Internet Protocol (IP). Internet telephony refers to communications services - voice, facsimile, and/or voice-messaging applications - that are transported via the Internet,

rather than the Public Switched Telephone Network (PSTN). The basic steps involved in originating an Internet telephone call are conversion of the analog voice signal to digital format and compression/translation of the signal into Internet protocol (IP) packets for transmission over the Internet; the process is reversed at the receiving end.

VoIP

Voice over Internet Protocol, or VoIP, uses the internet network based on packet-switching architecture using IP protocol. But since most telephony today is supported on PSTN (Public Switched Telephone Network) based on circuit-switched architecture, most VoIP based telephony uses a combination of the Internet and PSTN.

The diagram below shows the possible combinations in which VoIP currently operates



- ***What a VoIP call typically involves?***

Though the applications as discussed above can behave in different ways depending on the requirements, but at the fundamental level they are all same – One needs to capture the voice from microphone, process it (might involve encryption/compression/packetization) and send it across the network using TCP/IP protocol.

To setup a VoIP communication we need:

1. First the ADC (Analog to Digital Conversion) to convert analog voice to digital signals (bits)
2. Now the bits have to be compressed in a good format for transmission: there is a number of protocols we'll see after.
3. Here we have to insert our voice packets in data packets using a real-time protocol (typically RTP over UDP over IP)
4. We need a signaling protocol to call users: ITU-T H323 does that.
5. At Receiving end we have to disassemble packets, extract datas, then convert them to analog voice signals and send them to sound card (or phone)

All that must be done in a real time fashion cause we cannot waiting for too long for a vocal answer!

Steps 2 and 3 are in fact core of the application and involves more than what is just written above and involves many issues. The first major issue is of speed. This is the most critical factor in these kind of applications. In a very less amount of time your application should be able to do a lot of processing which involves peeking to get the compressed voice packet, decompress them (and decrypting them if encrypted) and then send them to sound card for playing. Also, at the same time the application should be able to do the reverse of the above like preparing and sending sound buffers to sound card for filling them with sound data, retrieve them, compress them (encrypt them too if required) and then send them to the remote machine. All this should be done in real time without delays. Thus, application should process very fast. As the bandwidth availability on normal telephone network (PSTN) is limited of the order of ~56 Kbps and that too not being consistent, a fair amount of processor time is involved in processing the code written for managing compression/decompression.

Having a fair idea of what going to be inside let's go through these steps one - by - one.

- ***Intranet Vs Internet***

Developing this kind of application for LAN is remarkably easy as compared to the application being developed for the Internet and therefore, it's makes sense to first develop the application for LAN without using Codecs/Encryption modules, testing it there extensively and once satisfied with the performance of the application only then injecting the code for managing compression/ decompression and encryption. On LAN you usually got the bandwidth of the order of ~10/100 Mbps which is very large compared to what we usually gets on the internet. Also, at this available bandwidth, the transmission of voice is instant and do not require CODEC implementation. Perhaps, implementing that will only deteriorate the voice quality in case of application being developed exclusively for LAN use.

Establishing a connection between two or more remote machines.

The foremost task, before one can do anything concerning voice is establishing a connection between two ends. This require some protocol suite that can be used for the Inter network communication over the Internet and thus come in picture - TCP/IP protocol suite.

TCP/IP uses a packet switching technique involving routing of small data packets called datagrams which contains the sender and receiver's IP addresses which are unique to the Internet. Since the method directly maps on to the hardware and still is independent of the hardware, the communication is adaptable to any hardware based network and is efficient. For applications which would require to recover from transmission errors and packet losses themselves, the method of reliable stream transport service is more applicable as it is more reliable than connectionless packet delivery system. Since the system uses IP address only, it is completely hardware independent.

Now VoIP doesn't use TCP because it is too heavy for real time applications, so instead a UDP (datagram) is used.

Secondly, UDP has no control over the order in which packets arrive at the destination or how long it takes them to get there (datagram concept). Both of these are very important to overall voice quality (how well you can understand what the other person is saying) and conversation quality (how easy it is to carry out a conversation). RTP solves the problem enabling the receiver to put the packets back into the correct order and not wait too long for packets that have either lost their way or are taking too long to arrive (we don't need every single voice packet, but we need a continuous flow of many of them and ordered).

For a complete description of RTP protocol and all its applications see relative RFCs 1889 and 1890.

There are also other protocols used in VoIP, like RSVP, that can manage Quality of Service (QoS). RSVP is a signaling protocol that requests a certain amount of bandwidth and latency in every network hop that supports it.

For detailed info about RSVP see the RFC 2205

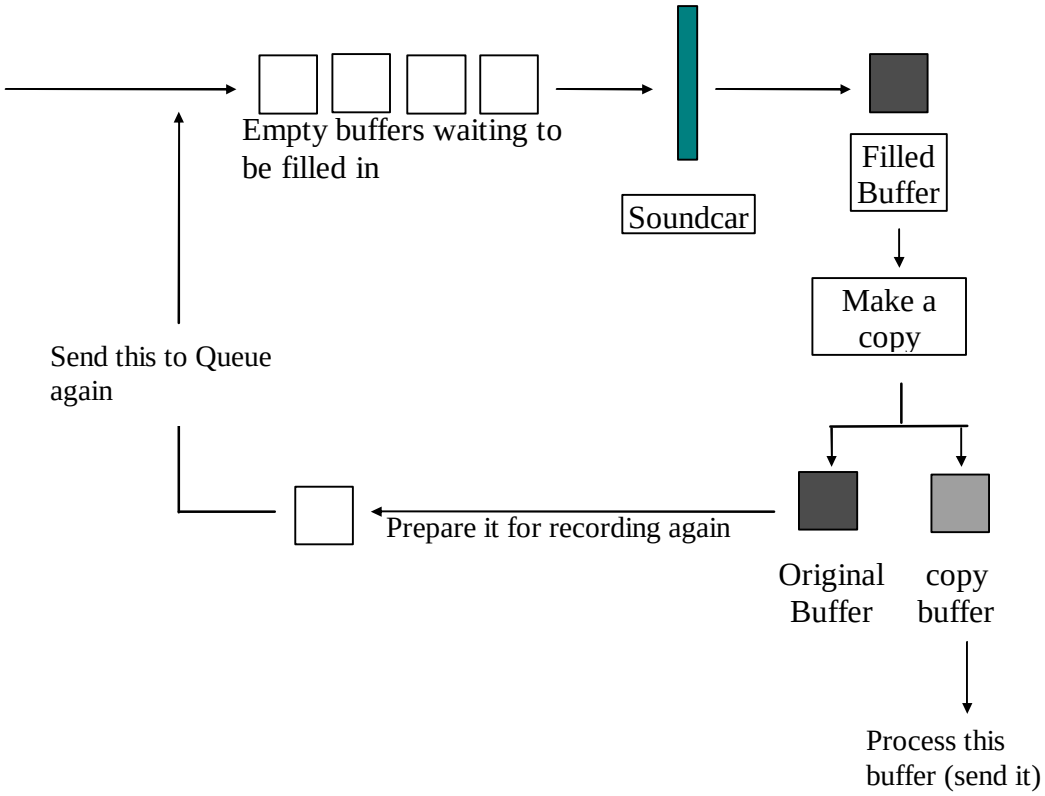
Following sections show some code for packetization, recording and playback of voice data using Win32 APIs.

Recording sound data and sending that to another machine.

This step can be subdivided into following sections

- Preparing buffers for recording voice data
- Sending these buffers in a queue to sound card
- Retrieve the filled buffers from the soundcard
- Process the data received in the above step
- Send the processed data to another machine

Graphically these steps can be depicted as :



Before moving further let's know about some important functions and structures related to Wave form audio. For detailed information on these please refer MSDN

Waveform Structures

1. WAVEFORMATEX
2. WAVEHDR

Waveform Functions

The major functions of our interest are :
Functions for wave recording (waveInXXX functions)
waveInOpen()
waveInPrepareHeader()
waveInAddBuffer()
waveInStart()
waveInUnprepareHeader
waveInClose()
waveInStop

Functions for playing wave data (waveOutXXX functions)
waveOutOpen()
waveOutClose()
waveOutPrepareHeader()
waveOutRestart()
waveOutUnprepareHeader()
waveOutWrite()

Preparing buffers for recording voice data

Before starting the process of voice recording and playing you have to declare and initialize the above described structures and handles related to wave input/output.

Two important waveform handles are :
HWAVEIN = This handle identifies the open waveform-audio input device.
HWAVEOUT = This handle identifies the open waveform-audio output device.

Let's define a structure for our convenience to be used in the code as :

```
typedef struct _LevelStreamData {
        LPWAVEHDR               alpWave[BUFFERSNO];
        WAVEFORMATEX  FAR       *pwf;
        HWAVEIN                 hwi;
        int                     buffersize;
}       LEVELSTREAMDATA, FAR * PLEVELSTREAMDATA;
```

BUFFERSNO are defined as

#define BUFFERSNO 4

Here you can define the total number of buffers that you want to use.

Now before starting the recording and playing of voice data you have to initialize these voice related structures and prepare the audio input and output device.

Inthe following lines of code we will see how one can open a wave device in the given format, queue up all the buffers and start data streaming.

step 1: Declare a member of PLEVELSTREAMDATA structure defined above.
PLEVELSTREAMDATA pInfo;

step 2: Initialize the above declared structure member with appropriate value like
pInfo->pwf->nChannels = 1;
pInfo->pwf->wBitsPerSample = 8;
pInfo->pwf->nSamplesPerSec = 11025;
pInfo->pwf->wFormatTag = WAVE_FORMAT_PCM;
pInfo->pwf->nBlockAlign = pInfo->pwf->nChannels * pInfo->pwf->wBitsPerSample / 8;
pInfo->pwf->nAvgBytesPerSec = pInfo->pwf->nSamplesPerSec * pInfo->pwf->nBlockAlign;
pInfo->buffersize = pInfo->pwf->nAvgBytesPerSec/UPDATES_PER_SEC;

Where UPDATES_PER_SEC is the part of a second for which you want to record the voice data and is defined as

#define UPDATES_PER_SEC 5
In which case your buffer size will record one fifth second's voice.

step 3: Opening the waveform input device (soundcard) for recording
The function for this goes like this :
waveInOpen(&pInfo->hwi, WAVE_MAPPER, (LPWAVEFORMATEX)pwf, (DWORD) hDlg, 0, CALL_BACK_WINDOW)

where hDlg is the handle to the window receiving messages.

step 4: Setting all the wave headers to NULL for cleanup
for(i=0; i<BUFFERSNO; i++)
pInfo->alpWave[i] = NULL;

step 5: Allocate, prepare and add all the buffers
for(i=0; i<BUFFERSNO; i++)
{

```
                pInfo->alpWave[i] = (LPWAVEHDR)
GlobalLock(GlobalAlloc(GMEM_MOVEABLE |          GMEM_SHARE,
sizeof(WAVEHDR) + pInfo->buffersize));

                if(pInfo->alpWave[i] == NULL)       return FALSE;   //Unable to allocate
memory

                pInfo->alpWave[i]->lpData = (LPSTR)(LPBYTE)(pInfo->alpWave[i] +
1);
                pInfo->alpWave[i]->dwBufferLength = pInfo->buffersize;
                pInfo->alpWave[i]->dwBytesRecorded = 0;
                pInfo->alpWave[i]->dwUser = 0;
                pInfo->alpWave[i]->dwFlags = 0;
                pInfo->alpWave[i]->dwLoops = 0;

                if(waveInPrepareHeader(pInfo->hwi, pInfo->alpWave[i],
sizeof(WAVEHDR)))                          return FALSE;
                //Above function prepare these buffers for waveform input finally

                if(waveInAddBuffer(pInfo->hwi, pInfo->alpWave[i],
sizeof(WAVEHDR)))
                        return FALSE;
                //Above function sends these buffers to the given waveform input device
                }
```

step 7: Finally starting the wave input

With the execution of waveInStart() function the waveform input device starts recording. The function is called like this.

```
int errorCode = waveInStart( pInfo->hwi);
if(errorCode) {
        MessageBox("Error in starting wave recording");
        return FALSE;
}
```

Retrieving filled buffers from the sound card

Once the wave recording is started, the recorded buffers from the sound card should be retrieved and processed . Once these filled buffers are retreived they are used for the following purpose :

- Sending the filled sound buffer to another machine.
- Sending this retrieved sound buffer again to the soundcard

Whenever an input buffer is full with the waveform-audio data and the buffer is being returned to the application the MM_WIM_DATA message is sent to a window.
This all can be accomplished by processing this window message

A code piece showing how this message can be processed is written here :

```
case MM_WIM_DATA:
        wRBuffer=(char*)((LPWAVEHDR)lParam)->lpData;
        _beginthread((void *)sendAThread,0,(void (__cdecl *)(void *))wRBuffer);

        //whereas wRBuffer is declared as :
        //char *wRBuffer;
        //and sendAThread is a function which receives the buffer as parameter and do all
        the processing
        //on it. Since the execution of code shouldn't affect the application, therefore it is
        called on //a separate thread.
        //The sendAThread function is declared like this :

        //      void sendAThread(char *sbuffer)
        //      {
        //              //code to compress the received buffer
        //              //and then sending it to another machine
        //              return;
        //      }

        waveInUnprepareHeader((HWAVEIN)wParam,(LPWAVEHDR)lParam,sizeof(
        WAVEHDR));

        //The waveInUnprepareHeader function cleans up the preparation performed by
        the
        //waveInPrepareHeader function. This function must be called after the device
        driver fills a //buffer and returns it to the application. You must call this function
        before freeing the buffer.

        waveInPrepareHeader((HWAVEIN)wParam,(LPWAVEHDR)lParam,sizeof(WA
        VEHDR));

        // The waveInPrepareHeader function prepares a buffer for waveform-audio
input.

        waveInAddBuffer((HWAVEIN)wParam,(LPWAVEHDR)lParam,sizeof(WAVE
        HDR));

        // The waveInAddBuffer function sends an input buffer to the given waveform-
        audio input //device. When the buffer is filled, the application is notified.
```

```
break;
```

Now done with recording and sending the voice data to another machine the task remained is to peek for the sound buffers sent from remote machine and than send the data received to sound card after proper processing of that.

Receiving a Voice packet from another remote machine and send it to soundcard for playing.

Before sending the received buffers to sound card for playing you had to do stuff like :

- Declaring format in which to play the voice data.
- Preparing buffers to play the voice data
- sending these buffers in a queue to sound card.
- Retrieve the Played buffers from the soundcard.
- Fill the retrived buffer again and send it to sound card

Declare and initialize a member of WAVEFORMATEX structure:

```
WAVEFORMATEX wfx

wfx.wFormatTag=WAVE_FORMAT_PCM;
wfx.nChannels=1;
wfx.wBitsPerSample=8;
wfx.nSamplesPerSec=11025;
wfx.nBlockAlign=wfx.nChannels*wfx.wBitsPerSample/8;
wfx.nAvgBytesPerSec=wfx.nSamplesPerSec*wfx.nBlockAlign;
wfx.cbSize=0;
```

Declare and initialize an array of LPWAVEHDR structure:

```
LPWAVEHDR        lpwhr[2];
for(int bon=0; bon<2; bon++)
{
lpwhr[bon]=(LPWAVEHDR)GlobalLock(GlobalAlloc(GMEM_MOVEABLE|G
MEM_SHARE,(DWORD)sizeof(WAVEHDR)));
lpwhr[bon]->lpData=(LPSTR)LocalAlloc(LMEM_FIXED,11025);
lpwhr[bon]->dwUser=0;
lpwhr[bon]->dwFlags=0;
lpwhr[bon]->dwLoops=0;
}
```

As the packet receiving part in itself is a resource consuming process, have a separate thread to do this job.

Following is a self explanatory skeleton of this function:

```
void ReceiveAndPlayThread()
{
// Peek for the incoming buffer and receive it
// if compressed then write code here for decompressing it
// Assign it to one of the array element of LPWAVEHDR structure declared above

Then prepare this waveform data block for playing
// waveOutPrepareHeader(hwo,lpwhr[TOGGLE],sizeof(WAVEHDR))

Send this data block to  the waveform output device using waveOutWrite()
// waveOutWrite(hwo,lpwhr[TOGGLE],sizeof(WAVEHDR));

// TOGGLE gives the number of the data block free for this operation
}
```

Whenever the buffers sent to the output device above are played or as the result of a call to the **waveOutReset** function, they are returned to the application and the MM_WOM_DONE message is sent to a window. This message can be processed to cleans up the preparation performed by the **waveOutPrepareHeader** function. This can be done as shown :

```
case MM_WOM_DONE:
waveOutUnprepareHeader((HWAVEOUT)wParam,(LPWAVEHDR)
lParam,sizeof(WAVEHDR));
break;
```

This was some code about how to record and playback the packetized voice data to help developers gather some insight into what actually is the core of VoIP applications.

• *VoIP Problems*

By using voice communication, near-perfect audio is the goal. After all, the end-users do not care how things are done. They just want to be able to have a good sound. Therefore, you need to be aware of the issues that affects voice quality and try to minimize the effect of these.

Four common problems that affect voice quality are:

- Delay
- Jitter
- Packet loss
- Sequence order problems

Delay

Excessive end-to-end delay makes conversation inconvenient and unnatural. Each component in the transmission path - sender, network, and receiver - adds delay. ITU-T G.114 (One-Way Transmission Time) recommends 150 mSec as the maximum desired one-way latency to achieve high-quality voice.

Jitter

Quantifies the effects of network delays on packet arrivals at the receiver. Packets transmitted at equal intervals from the left gateway arrive at the right gateway at irregular intervals. Excessive jitter makes speech choppy and difficult to understand. Jitter is calculated based on the inter-arrival time of successive packets. For high-quality voice, the average inter-arrival time at the receiver should be nearly equal to the inter-packet gaps at the transmitter and the standard deviation should be low. Jitter buffers (packet buffers that hold incoming packets for a specified amount of time) are used to counteract the effects of network fluctuations and create a smooth packet flow at the receiving end.

Packet loss

Typically occurs either in bursts or periodically due to a consistently congested network. Periodic loss in excess of 5-10% of all voice packets transmitted can degrade voice quality significantly. Occasional bursts of packet loss can also make conversation difficult.

Sequence Errors

Congestion in packet switched networks can cause packets to take different routes to reach the same destination. Packets may arrive out of order resulting in garbled speech.

Conclusion

To conclude, VoIP is here to stay !. It threatens to replace the existing POTS completely. The bandwidth is becoming cheaper with every passing day, and it is this reason which is acting as a main proponent of VoIP. The combination of voice, video and data will become a commodity for almost everybody, like the telephone that we are have been using for a century.

VoIP services will include:

- Multimedia conferencing -- allowing Multiple users to communicate by voice and video
- Multicasting - disseminating voice, video, and/or data to a large targeted group

- Collaborative workgroup applications - facilitating multiple users to interact verbally and visually while ;haring access to common data and applications
- Call center applications - using IP telephony to enhance today's Web-based e-commerce through interaction with live service representatives
- Unified messaging - consolidating email, pager, voicemail, and fax services into a single IP-based service
- IP call waiting - expanding the number and type of simultaneous incoming calls beyond the two voice calls that the PSTN currently supports

Few links of interest are:

http://www.packetizer.com/
http://www.protocols.com/papers/voip.htm
http://www.cis.ohio-state.edu/~jain/refs/ref_voip.htm

Mindfire Solutions is an IT and software services company in India, providing expert capabilities to the global market. Mindfire possesses experience in multiple platforms, and has built a strong track record of delivery. Our continued focus on fundamental strengths in computer science has led to a unique technology-led position in software services.

To explore the potential of working with Mindfire, please drop us an email at info@mindfiresolutions.com. We will be glad to talk with you.

To know more about Mindfire Solutions, please visit us on www.mindfiresolutions.com

---