DESIGN AND DEVELOPMENT OF AN INTERNET TELEPHONY TEST DEVICE

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

TURGUT ÇELİKADAM

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

IN

THE DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

DECEMBER 2003

Approval of the Graduate School of Natural and Applied Sciences

Prof. Dr. Canan Özgen
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. Mübeccel DEMİREKLER
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Prof. Dr. Faruk Rüyal ERGÜL
Supervisor

Examining Committee Members

Prof. Dr. Mete SEVERCAN	
Prof. Dr. Faruk Rüyal ERGÜL	
Prof. Dr. Yalçın TANIK	
Prof. Dr Hasan GÜRAN	
Serkan SEVİM	

ABSTRACT

Design and Development of an Internet Telephony Test Device

Çelikadam, Turgut

M.S., Department of Electrical and Electronics Engineering

Supervisor : Prof. Dr. F. Rüyal Ergül

December 2003, 90 pages

The issues involved in Internet telephony (Voice over Internet Protocol (VoIP) device) can be best understood by actually implementing a VoIP device and studying its performance. In this regard, an Internet telephony device, providing full duplex voice communication over internet, and a user interface program have been developed. In the process, a number of implementation issues came into focus, which we have touched upon in this thesis.

Transport layer network protocols are discussed in the concept of real time streaming applications and Real Time Protocol (RTP) is modified to use as transport layer protocol in developed VoIP device. Adaptive playout buffering algorithms are studied and compared with each other by trace driven simulation experiments with objective measures. A method to solve clock synchronization problem in streaming internet applications is presented.

One way and round trip delay measurement functionalities are added to the VoIP device, so that device can be used to investigate the network characteristics.

Keywords: VoIP, Internet Telephony, Adaptive Playout Buffering, Real Time Protocol

ÖZ

İnternet Telefonu Test Cihazı

Tasarımı ve Geliştirilmesi

Çelikadam, Turgut

Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü

Tez Yöneticisi : Prof. Dr. F. Rüyal ERGÜL

Aralık 2003, 90 sayfa

Internet telefonu (Internet Protokolü Üzerinden Ses (IPÜS) cihazı) tarafından kapsanan önemli noktalar, en iyi şekilde bir IPÜS cihazı gereçekleyerek ve cihazın performansı üzerinde çalışarak en iyi şekilde anlaşılabilir. Bu göz öünde bulundurularak, Internet üzerinden çift yönlü ses iletişimine olanak sağlayan Internet telefonu ve kullanıcı arayüz proğramı geliştirilmiştir. Bu süreçte, gerçekleme ile ilgili birçok önemli nokta ilgi odağı olmuş ve bunlara bu tez kapsamında değinilmiştir.

Taşıma katmanı ağ protokolleri gerçek zamanlı akan uygulamalar kapsamında tartışılmış ve Gerçek Zaman Protokolü (GZP) geliştirilen IPÜS cihazında taşıma katmanı protokolü olarak kullanılmak üzere değiştirilmiştir. Uyarlamalı çalma tampon algoritmaları üzerinde çalışılmış ve bu algoritmalar birbirleriyle simülasyon deneyleri yaparak, nesnel ölçütlerle karşılaştırılmıştır. Akan internet uygulamalarında saat senkronizasyon problemini çözmek için bir yöntem sunulmuştur.

Tek yönlü ve dairesel döngü gecikme ölçüm özellikleri IPÜS cihazına eklenmiş, böylelikle ağ karakteristiklerini araştırmada kullanılabilmesine imkan sağlanmıştır.

Anahtar Kelimeler : IPÜS, Internet telefonu, Uyarlamalı çalma tamponu, Gerçek Zaman Protokolü

To My Family

ACKNOWLEDGEMENTS

I wish to express my sincere gratitude to Prof. Dr. Faruk Rüyal Ergül for his supervision, valuable guidance and helpful suggestions.

I would like to express my sincere appreciation to Serkan Sevim, K. Gökhan Tekin, Mehmet Karakaş, Hasan Çitçi in ASELSAN Inc. for their valuable friendship, help and support. I am also grateful to ASELSAN Inc. for the facilities provided for the completion of this thesis.

TABLE OF CONTENTS

ABSTRACT .... **iii**

OZ .... **v**

ACKNOWLEDGEMENTS .... **viii**

TABLE OF CONTENTS .... **ix**

LIST OF TABLES .... **xi**

LIST OF FIGURES .... **xii**

CHAPTER

1. INTRODUCTION **1**

2. PLAYOUT PROBLEM **6**

2.1 Introduction. 6

2.2 Playout problem 6

2.3 Adaptive Playout Algorithms 8

2.3.1 Illustration of the adaptive playout 11

2.3.2 Algorithm 1 13

2.3.3 Algorithm 2 14

2.3.4 Algorithm 3 15

2.3.5 Algorithm 4 15

3.PROTOCOLS **18**

3.1 Ethernet 18

3.2 Internet Protocol (IP) 20

3.3. Real Time Protocol 23

4. SYSTEM ARCHITECTURE AND IMPLEMENTATION **28**

4.1 Hardware 30

4.1.1 Voice Digitization 33
4.1.2 CODEC- FPGA Serial Interface 35
4.1.5 Microprocessor-FPGA interface: 39
4.2 Microprocessor Software 43
4.2.1 Control Functions. 43
4.2.1.1 Network Settings 45
4.2.1.2 Clock Offset Calculation 46
4.2.1.3 Connection Establishment 50
4.2.1.4 RTT Measurement 51
4.2.2 Sender and Receiver Functions 53
4.2.2.1 Sender Functions 54
4.2.2.2 Receiver Functions 56
4.3 User Interface Program 61
4.4 Total End to End Delay 64
4.5 VoIP Device Compatible Computer Software Development 65
4.5.1 The basic system design 66
4.5.2 Control Module 67
4.5.3 Receiver Module 69
4.5.4 Transmitter Module 71
5. ADAPTIVE PLAYOUT ALGORITHM SIMULATIONS AND COMPARISONS 73
5.1 Simulations 74
5.2 Comparison and Discussion of results 81
6. CONCLUSION AND FUTURE WORK 84
REFERENCES 87
APPENDIX
A. Pseudo code for the *Send_Adc_Data_Task* 89

LIST OF TABLES

TABLE

4.1 DPRAM data format…………………………... …………………………...41

4.2 DPRAM Sections…………………………………………………………42

4.3 UART Commands………………………………………………………...44

4.4 Packet Type Indicator Characters………………………………………45

5.1 Mean Playout Delay and Number of Lost Packets for Trace 1………………...78

5.2 Mean Playout Delay and Number of Lost Packets for Trace 2………………...81

LIST OF FIGURES

FIGURE

2.1 Generation and reconstruction of Packetized voice ……………………….…7

2.2 Timing associated with packet *i*……………………………………….......9

2.3 Ilustration of playout mechanism……………………………………........12

2.4 Pseudo code of Algorithm 2………………………………………….......14

2.5 A typical Delay Spike………………………………………...……..15

2.6 Algorithm 4 (Spike detection Algorithm)…………………………...…...16

3.1 Bus based broadcast network……………………………………………..19

3.2 Ethernet Frame Format…………………………………………….......19

3.3 IP Header………………………………………………………….....21

3.4 RTP Header……………………………………………………..……25

3.5 UDP Header……………………………………………….... 26

3.6 Packet Nesting………………………………………………… …......27

4.1 VoIP Board…………………………………………………….. …......29

4.2 VoIP Device…………………………………………………... 29

4.3 Block diagram of the VOIP board…………………………….… 32

4.4 Functional block diagram of the CODEC…………………………..33

4.5 Timing diagram of the Codec serial interface……………………........35

4.6 CODEC serial interface frame format…………………………….....35

4.7 FPGA-ADPCM Processor Interface…………………………………...38

4.8 DPRAM- Microprocessor Interface……………………………………...41

4.9 Clock offset and skew between two clocks……………………………47

4.10 UDP packet exchange for clock offset calculation between clients A and B....48

4.11 UDP Packet’s Used for Clock Offset Calculation Between clients A and B…49
4.12 Voice UDP Packet Data section For PCM mode……………………………..54
4.13 Voice UDP Packet Data section for ADPCM mode…………………………54
4.14 DPRAM DAC Section Implemented as a Circular Buffer….………………..59
4.15 User Interface Program……………………………………………………..61
4.16 Two communicating VoIP Device……………………………………………62
4.17 Basic system structure of the VoIP software on a PC platform.......................66
4.18 Code segment to create sockaddr_in structure..67
4.19 Code segment to set the soundcard waveout properties...................................69
5.1 Delay Measurement Result for Trace 1………………………………………..75
5.2 PDF of Trace1 and Gamma Distribution Fitted to Trace 1……………………75
5.3 Calculated Playout Time by Algorithm 1 for Trace 1……..……………………76
5.4 Calculated Playout Time by Algorithm 2 for Trace 1………..…………………76
5.5 Calculated Playout Time by Algorithm 3 for Trace 1………..…………………77
5.6 Calculated Playout Time by Algorithm 4 for Trace 1………..…………………77
5.7 Delay Measurement Result for Trace 2………………..……………………...78
5.8 PDF of Trace1 and Gamma Distribution Fitted to Trace 2……………………79
5.9 Calculated Playout Time by Algorithm 1 for Trace 2…………………………79
5.10 Calculated Playout Time by Algorithm 2 for Trace 2..………………………80
5.11 Calculated Playout Time by Algorithm 3 for Trace 2..………………………80
5.12 Calculated Playout Time by Algorithm 4 for Trace 2..………………………81

CHAPTER 1

INTRODUCTION

In the past few years we have witnessed a significant growth in the Internet in terms of the number of hosts, users, and applications. The success in coping with the fast growth of the Internet rests on the Internet Protocol (IP) architecture's robustness, flexibility, and ability to scale.

By the availability of high bandwidth, new applications, such as Internet telephony also known as Voice over IP (VoIP), audio and video streaming services, video-on-demand, and distributed interactive games, have proliferated. These new applications have diverse quality-of-service (QoS) requirements that are significantly different from traditional best-effort service. For example, high-quality video applications, such as remote medical diagnosis and video-on-demand applications, demand reliable and timely delivery of high bandwidth data, and require QoS guarantees from the network.

On the other hand, a majority of multimedia applications including Internet telephony, video conferencing, and web TV, do not need in-order, reliable delivery of packets, and can tolerate a small fraction of packets that are either lost or highly delayed, while still maintaining reasonably good quality. These applications employ end-to-end control that adapts to the changing dynamics of the network and can often deliver the acceptable quality to users.

VoIP, which is the subject of this thesis, is the transportation of speech signals in an acceptable method from sender to destination over an Internet network. The speech signal is digitized pieces of voice conversation sampled at regular intervals. These samples are sent via the network to the desired destination where they are reconstructed into an analog signal representing the original voice. Packet network based voice is extremely desirable due to advantages like cost effectiveness and easy integration with other information channels. This can lead to single network that provides all services.

Unlike conventional telephony, VoIP is afflicted with problems that affect its quality, like delay, jitter and loss. High quality voice communication over the Internet requires low end-to-end delay and low loss rate, [1]. However, best effort networks such as today's Internet, are characterized by highly varying delay and loss characteristics that contradict with QoS requirements. Both delay and loss result from buffering within the network. As packets traverse the network, they are queued in buffers (adding to their end to end delay) and from time to time dropped due to buffer overflow. A detailed work on end to end Internet packet dynamics can be found in [10]. A number of playout adaptation and loss recovery techniques exist to counter these problems, respectively. Correlation between delay and loss is discussed in [1].

Compensating for loss using end-to-end protocols and algorithms can be done using a number of mechanisms, including local repair (interpolation of missing data using the surrounding packets) and interleaving, [2]. There has been much interest in the use of packet level Forward Error Correction (FEC) mechanisms. All of the FEC mechanisms send some redundant information, which is based on previously transmitted packets. Waiting for the redundant information results in a delay penalty, and consequently an increase in size of the playout buffers. When network loss rates are high, accepting the delay penalty for increased recovery capabilities is appropriate. However when network loss rates are low, The FEC may not provide useful information, and increasing the playout buffer sizing to wait for it is not appropriate. In this thesis, interpolation by zero insertion method is used to compensate for packet loss and FEC methods are left as an future scope of work. Detailed information about the usage of FEC with adaptive playout algorithms can be found in [2].

In real-time applications, such as VoIP, a smoothing buffer is typically used at a client host to compensate for variable delays (delay jitter). Received packets are first queued into the smoothing buffer. After several packets are queued, actual decoding is started. Then, the consequences of the delay variations within the network can be minimized (We refer to this delay as the playout delay). Choosing the playout delay is important because it directly affects the communication quality of the application; if the playout delay is set to be too short, the client application treats packets to be lost even if those packets eventually arrive. On the contrary, the large playout delay may introduce an unacceptable delay that the client users cannot be tolerant. The packet transmission delay between the server and client may be varied according to the network condition in the Internet, and hence, the adequate playout delay is heavily dependent on variations of packet transmission delays. That is, a difficulty exists in determining the playout delay. Adaptive playout delay estimation algorithms are used to compensate the delay variations. These algorithms adapts to the changing dynamics of the network by estimating the network delay and delay variance, [3]. According to the estimated variables of delay and delay variance playout delay is set. Changing the playout delay in between an audio stream will cause jitter in the played out speech. Thus, adjustments are made only in the silence periods between two talkspurts in the audio stream. Therefore, a mechanism for Voice Activity Detection (VAD) is needed to discriminate the silence periods. Enabling the VAD and not transmitting packets in the silence periods can reduce transmission rate. Since one communication party is speaking usually other is not; VAD can reduce transmission rate %50.

Also, when using a data network for real-time voice communication, there comes a question of which transport layer protocol should be used? This protocol should introduce minimum delay and overhead. Transport layer protocols such as Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) can be used. TCP/IP is considered to be "reliable". Reliable means that each individual packet that is sent over the network is verified at the receiver and acknowledged. In TCP, a retransmission mechanism exist that every unacknowledged packet is retransmitted. Therefore, TCP guarantees packet delivery. This retransmission mechanism introduces a extra delay on packet delivery therefore unacceptable. In contrast, UDP has no retransmission mechanism. While this reduces the overhead and delay in processing, packets can arrive out of order or be dropped from reception completely. The latest IP protocol developed specifically for streaming audio and video over the Internet is Real-Time Transfer Protocol (RTP). It is described in RFC 1889,[4].RTP imposes packet sequencing and time stamping on a UDP data stream to ensure sequential packet reconstruction at the receiver while not imposing the high processing overhead of reliable transmission. In the concept of this thesis a RTP like custom protocol is developed that is used as Transport Layer Protocol for the developed VoIP device.

The issues involved in VoIP can be best understood by actually implementing a VoIP device and studying its performance. In this regard, we have built a VoIP device. In the process, a number of implementation issues came into focus, which we have touched upon in this thesis. Adaptive playout algorithms are compared using the objective measures, in trace driven simulation experiments where traces are collected with the developed VoIP device. An adaptive playout algorithm was chosen according to the simulation results and incorporated into the developed VoIP device to compensate for delay jitter. Network delay, used in playout adaptation algorithms, is measured using both sender and receiver clock. This causes clock synchronization problem. This problem is solved by estimating the clock offset with the method given in [5]. Loss packets are interpolated by zero insertion to compensate for loss packets. An experimentally developed VAD mechanism is implemented to discriminate talkspurts and to be able to set the playout delay at the start of the each talkspurt and to reduce the transmission rate. Adaptive Differential Pulse Code Modulation (ADPCM) is used to further reduce the transmission rate. RTP wise custom protocol is developed and used in the design as the transport layer protocol. A user interface program is developed that runs on windows based PC. This software is used both to control the device and to collect running measurement results of network delay.

Organization of the thesis is as follows. In Chapter 2 playout problem is discussed and four adaptive playout algorithms, given in [3], are explained. In Chapter 3, network protocols used in the VoIP device and custom RTP protocol are discussed and explained. In Chapter 4, device architecture and implementation details of the VoIP device and user interface program are described. In chapter 5, the trace driven simulation results of the four adaptive playout algorithms that will be explained in Chapter 2 are given. These algorithms are compared according to the simulation results. The thesis ends in Chapter 6 with a summary and a discussion about the further scope for work.

CHAPTER 2

PLAYOUT PROBLEM

2.1 Introduction.

Streaming systems rely on buffering at the client to protect against the random packet losses and delays that characterize a best-effort network. These parameters vary depending on the locations of the senders and the receivers; with typical loss rates of 0-20% and one-way delays of 5-500 ms, [6]. Buffering reduces a system's sensitivity to short-term fluctuations in the data arrival rate by absorbing variations in end-to-end delay that is called jitter. However, buffering has drawbacks. While the amount of protection a buffer offers grows with its size, so does the latency that it introduces. Unfortunately, this additional delay lowers the perceived QoS. In streams of live events or in two-way communication, latency is noticeable throughout the session,[3]. On the other hand, if the playout delay is set to low, the network-introduced delay will cause some packets to arrive too late for playout and thus be lost, which also lowers the perceived QoS. The main objective of jitter buffering is to keep the packet loss rate under 5% and to keep the end-to-end delay as small as possible, [3].

2.2 Playout problem

We shall first discuss the underlying model for packetized voice, the assumptions regarding their generation and the mechanism for their playout.

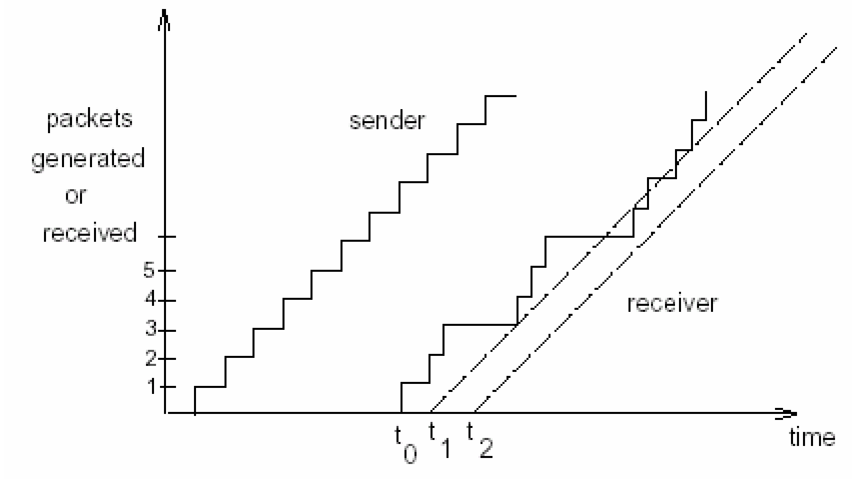

Figure 2.1 : Generation and reconstruction of Packetized voice

Figure 2.1 shows the operation of the sending and receiving hosts while taking part in an audio session. At the sender, packets are periodically generated as a result of the periodic sampling of an audio source. When the audio source is active (i.e., sound is being produced) packets containing the audio samples are generated and sent into the network. The staircase nature of the sender in Figure 2.1 indicates that packets are being generated periodically at the source. One commonly used standard is sending one 160 byte audio packet, which is generated approximately every 20 milliseconds when the speaker is in *talkspurt*, i.e. when there is voice activity. The average talkspurt length is typically on the order of several 100's of milliseconds, although the lengths can vary with different silence detection thresholds.

Packets incur random delays while traversing the network. This is illustrated by the decidedly non-staircase nature of the number of received packets as a function of time in Figure 2.1. In order to smooth out such delay jitter, a receiving host can delay the initiation of periodic playout of received packets for some time. For example, in Figure 2.1, if the receiver delays the beginning of playout until t_2, all packets will have been received by the time their playout is scheduled. The 45 degree line emanating from t_2 indicates the playout time of packet *i* under a periodic playout strategy which begins playout at *i*. On the other hand, if the playout delay begins at t_1, there is a shorter playout delay, but packets 6, 7, and 8 will be lost at the receiver, having arrived after their scheduled playout time. This illustrates the tradeoff between the delay that an audio application is willing to tolerate and the packet loss suffered as a result of the late arrival of packets.

Figure 2.1 depicts a playout strategy in which the playout delay is fixed, called the *fixed playout scheme*. If both the propagation delay and the distribution of the variable component of network delay are known, a fixed playout delay can be computed such that no more than a given fraction of arriving packets are lost due to late arrival, [3]. One problem with this approach is that the propagation delay is not known (although it can be estimated and typically remains fixed throughout the duration of the audio call). A more serious concern is that the end-to-end delay distribution of packets within a talkspurt is not known, and can change over relatively short time scales.

An approach to dealing with the unknown nature of the delay distribution is to estimate these delays and adaptively respond to their change by dynamically adjusting the playout delay. In the following section, we shall describe a four receiver-based algorithms, presented in [3], for performing such delay estimation and dynamic playout delay adaptation. As we will see, these algorithms determine the playout delay on a per-talkspurt basis. Within a talkspurt, packets are played out in a periodic manner, thus reproducing their periodic generation at the source. However, the algorithms may change the playout delay from one talkspurt to the next, and thus the silence periods between two talkspurts at the receiver may be artificially elongated or compressed (with respect to the original length of the corresponding silence period at the sender). Compression or expansion of silence by a small amount is not noticeable in the played-out speech.

2.3 Adaptive Playout Algorithms

In this section we define four adaptive playout delay adjustment algorithms. In describing these algorithms the notation in Figure 2.2 will be useful. Figure 2.2 shows the various times associated with the sending and receiving of packet i within an audio call, [3].

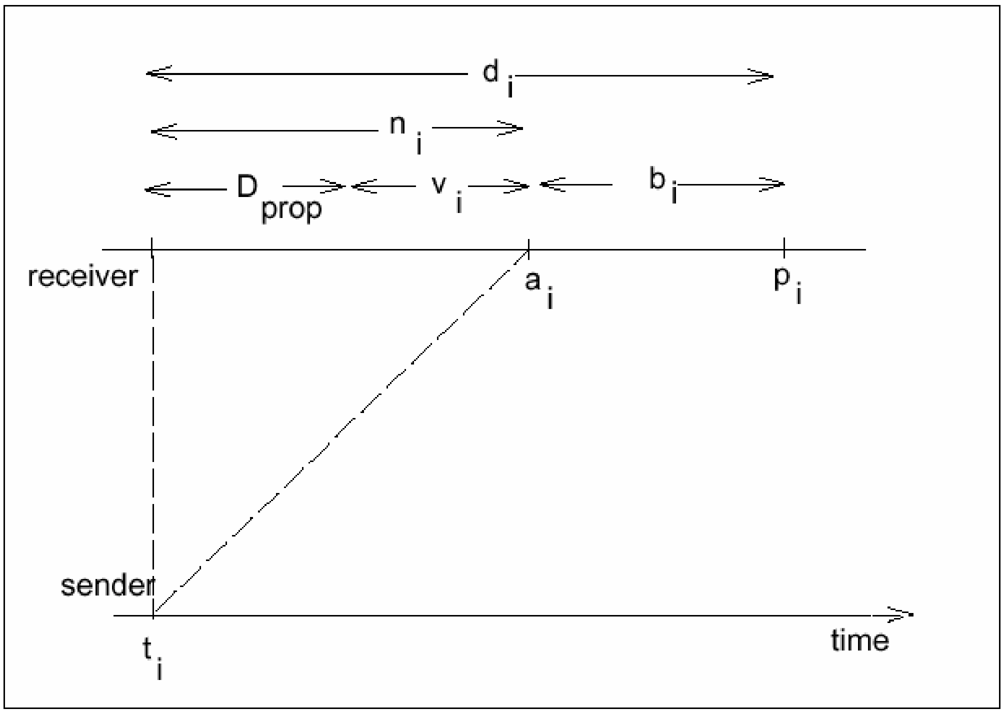

Figure 2.2: Timing associated with packet *i*

The following times associated with packet *i* are introduced, in accordance with Figure 2.2:

t_i : the time at which packet *i* is generated at the sending host,

a_i: the time at which packet *i* is received at the receiving host,

pi: the time at which packet *i* is played out at the receiving host,

D_{prop}: the propagation delay from the sender to the receiver, which is assumed to be constant throughout the lifetime of an audio connection,

v_i: the queuing delay experienced by packet i as it is sent from the source to the destination host,

b_i : the amount of time that packet i spends in the buffer at the receiver awaiting its scheduled play out time, $b_i = p_i - a_i$,

d_i : the amount of time from when the i^{th} packet is generated by the source until it is played out at the destination host, $d_i = p_i - t_i$, this will be referred to as the playout delay of packet i,

n_i : the total delay introduced by the network. $n_i = a_i - t_i$.

In determining the playout point for packet i, two cases are considered depending on whether or not it is the first packet in a talkspurt,[3]. If packet i is the first packet of a talkspurt, its playout time p_i is computed as:

$$p_i = t_i + d_i' + 4 \times v_i' \qquad (2.1)$$

Where d_i' and v_i' are estimates of the mean and variation in the end to end delay during the talkspurt. The playout point for any subsequent packet in a talkspurt is computed as an offset from the point in time when the first packet in that talkspurt was played out. If i was the first packet in a talkspurt and packet j belongs to this talkspurt the playout point for j is computed as:

$$p_j = p_i + t_j - t_i$$

We note that d_i' and v_i' are computed for every packet received although they are only used to determine the playout point for the first packet in any talkspurt. The four algorithms described in section 2.3.1 through 2.3.1.4 differ only in the manner in which d_i' is computed. The computation of v_i' which in turn depends on d_i' is the same for all the algorithms and is defined in equation 2.1. From an intuitive standpoint the term 4x v_i' is used to set the playout time to be far enough beyond the delay estimate so that only a small fraction of the arriving packets should be lost due to late arrival A discussion of this variation measure and standard measures of variation such as standard deviation can be found in, [7].

2.3.1 Illustration of the adaptive playout

The playout mechanism is further illustrated in Figure 2.3. The graph at the top of the figure labeled A represents the network delay n_i on the y-axis experienced by packet *i* transmitted at time t_i.. Note that the unit of time on the x-axis is the inter packetization interval which is 16 ms in the case of our audio experiments. Two talkspurts are shown in the Figure 2.2, one starting at t_1=1 and another starting at t_7=9. The time axis labeled B shows the arrival pattern of the packets at the receiver. For example packets 2, 3 and 4 shown on top of each other arrive almost simultaneously at a_i = 8, as they experience different network delays. The remaining three axes illustrate the playout behavior for three possible delay adaptation scenarios.

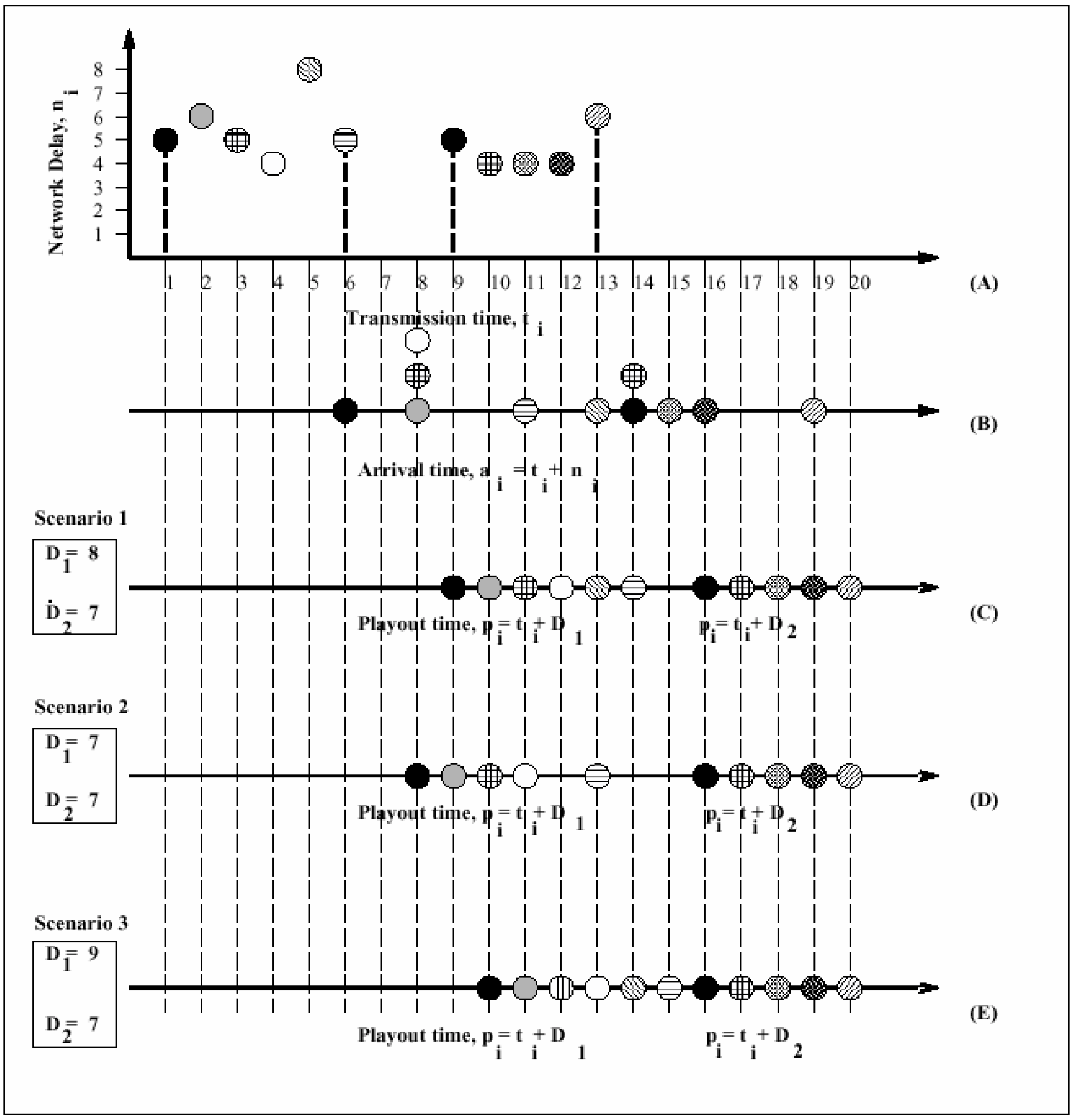

Figure 2.3: Ilustration of playout mechanism.

The axis labeled C computes the playout delay for talkspurt 1 to be 8 units and thus schedules the playing of packet 1 at time p_1=9 units. The remaining packets in talkspurt 1 are scheduled one after another in the order in which they were generated. In this example, it is then determined that the playout delay for the second talkspurt should be 7 units. Recall that this playout delay for the packet at the beginning of every talkspurt depends on the di' and the vi', which are computed for every packet seen so far, and which in turn depend on the delay adaptation algorithm used.

The axis labeled D illustrates a second possible playout scenario, in which playout delay for the first talkspurt is determined to be 7 units Note that this leads to the dropping of packet number 5 as it doesn't arrive at the receiver until after its scheduled playout time The axis labeled E shows yet another scenario in which the playout delay for talkspurt 1 is determined to be 9 units.

It is important to note how the silence period between the two talkspurts differs in these scenarios. In scenario 1, the silence period is one unit of time shorter than what was generated by the audio source; in the third scenario, the silence period is completely eliminated. From Figure 2.3, it is also clear that if we set the playout delay of a talkspurt to be greater than or equal to the maximum network delay experienced by any packet in that talkspurt, we would not have any late packet loss. Of course this value in not known a priori, although one could possible set a playout delay to a large enough value to ensure that a significant percentage of packets would not be lost. On the other hand, setting the playout delay to a high value leads to longer delays between the transmission and the playout of the audio packets; long delays are intolerable with interactive audio. Thus we desire a playout adaptation mechanism, which has low loss rate as well as low playout delay

We now describe the four playout adaptation algorithms presented in [3].

2.3.2 Algorithm 1

In first algorithm, the delay estimate for the i^{th} packet is calculated based on the RFC793 algorithm [8] and a measure of the variation in the delays is calculated as suggested by Van Jacobson [7] in the calculation of round trip time estimates for the TCP retransmit timer. Specifically the delay estimate for packet i is computed as

$d_i' = \alpha * d_{i-1}' + (1-\alpha) * n_i$

and the variation is computed as

$$v_i' = \alpha * v_{i-1}' + (1-\alpha) * |d_i - ni| \tag{2.2}$$

This algorithm is basically a linear recursive filter and is characterized by the weighting factor α.

2.3.3 Algorithm 2

The second algorithm is a small modification to the first algorithm based on a suggestion by Mills [9].

The idea is to use a different weighting mechanism by choosing two values of the weighting factor, one for increasing trends in the delay and one for decreasing trends. Variation estimate is calculated as in algorithm 1. The delay estimation algorithm is given in Figure 2.4.

```
If (ni>di') then
    $di' = \beta * di' + (1-\beta) * ni$
else
    $di' = \alpha * di' + (1-\alpha) * ni$
```

Figure 2.4: Pseudo code of Algorithm 2

2.3.4 Algorithm 3

Let *Si* be the set of all packets received during the talkspurt prior to the one initiated by *i*. The delay estimate is computed as;

$$d_i' = \min_{j \in S_i} \{n_j\} \qquad (2.3)$$

The delay variance is calculated as in algorithm 1.

2.3.5 Algorithm 4

Delay spikes are a common occurrence in the Internet. A spike constitutes a sudden, large increase in the end-to-end network delay, followed by a series of packets arriving almost simultaneously, leading to the completion of the spike. Figure 2.5, shows a typical spike; with the arrival time indicated on the x-axis, and the network delay experienced on the y-axis.

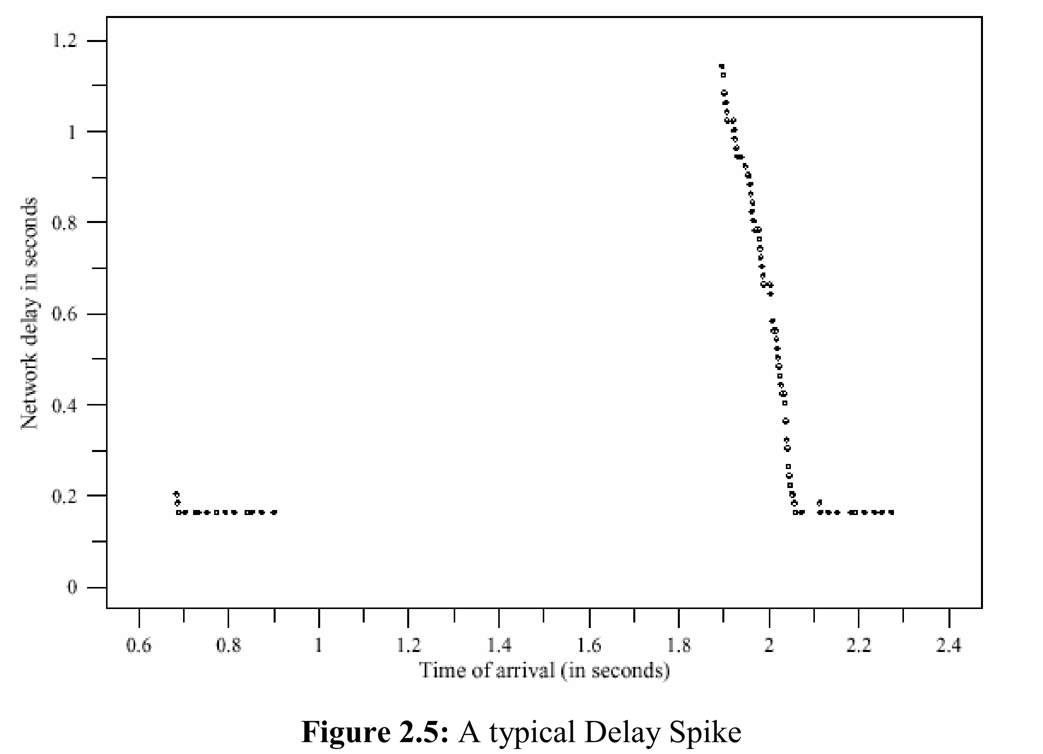

Figure 2.5: A typical Delay Spike

The algorithms discussed until now do not adapt fast enough to such spikes, taking too long to increase their delay estimate on detection of a spike and too long again to decrease their estimate once the spike is over. In [3], an algorithm is described to adapt to such spikes. This algorithm is called spike detection algorithm and it is given in Figure 2.6.

```
1. $n_i$=Receiver_timestamp –Sender_timestamp
2. if (mode==Normal  {
   if (abs($n_i - n_{i-1}$) > abs(v')*2+800
      var = 0 ;  /* Detected Beginnig of a spike*/
      mode = IMPULSE ;  }
   Else {
      var = var/2 +abs (( 2$n_i$-$n_{i-1}$-$n_{i-2}$) / 8);
      if var (<=63){
         mode = NORMAL; /*End of a spike*/
       $n_{i-2} = n_{i-1}$;
      $n_{i-1} = n_i$;
      Return;}
       }
3. if mode == NORMAL) then
      $d_i' = 0.125*n_i +0.875* d_{i-1}'$;
   Else
      $d_i' = d_{i-1}' +  n_i-n_{i-1}$ ;
      $v_i' = 0.125*abs(n_i - d_i') + 0.875 * v_{i-1}$;
4.  $n_{i-2} = n_{i-1}$;
    $n_{i-1} = n_i$;
    Return;
```

Figure 2.6: Algorithm 4 (Spike detection Algorithm)

Detection of the beginning of a spike is done by checking if the delay between consecutive packets at the receiver is large enough for it to be called a spike. On entering the impulse mode on detection of a spike, the spike is “followed”; in the sense that, the estimate is dictated only by the most recently observed delay values.

The detection of the completion of a spike is a bit difficult. The delay on completion of the spike could be different from the delay before the beginning of the spike. Nonetheless, one prominent characteristic is that a series of packets would arrive one after another almost simultaneously at the receiver, and almost immediately following the observed increase (upward spike) in delay. Since the packets within a talkspurt are transmitted at regular intervals at the sender, near simultaneous arrivals implies that subsequent packets in the burst of arrivals have experienced progressively smaller end-to-end network delays. So a variable is employed that keeps track of the slope of spike, which is indicated by *var* in algorithm 4 (Spike detection algorithm). When this variable has a small enough value, indicating that there is no longer a significant slope, the algorithm reverts back to normal mode.

One limitation of this algorithm is that the parameters involved in the detection of the beginning and end of a spike are dependent on the nature of spikes being observed.

These algorithms will be compared with each other with the trace driven simulations carried on MATLAB, to decide which algorithm to use in the developed VoIP device. Simulation traces are obtained using the developed VoIP device before actually implementing the adaptive algorithm chosen. Simulation method and simulation results are discussed in Chapter 5.

CHAPTER 3

PROTOCOLS

Using an Internet protocol network requires the utilization of an IP protocol for transmitting the information. Main functions of the IP are switching and routing. Networks are sometimes referred to as “packet” networks, since they communicate through the sending and receiving of data packets with known formats.

In the next sections, used protocols by the developed VoIP device are explained to form a background for chapter 4, where device architecture is explained. Since the device can be used in an Ethernet network, Ethernet protocol is mentioned. IP protocol and real time protocol, developed in the scope of this thesis, are explained.

3.1 Ethernet

Internet is designed for wide area networking. However, many companies, universities, and other organizations have large number of computers that must be connected. This need gave rise to the Local Area Network (LAN). Then, LANs can be connected to the Wide Area Networks (WANs), such as Internet, using gateways. In this section we will say a little bit about most popular LAN, Ethernet that is the network interface of the developed VoIP device.

Ethernet is bus based broadcast network usually operating at 10 Mbps to 10 Gbps, [4]. In the VoIP device 10 Mbps Ethernet interface is implemented. Computers or devices on Ethernet can transmit whenever they want to; if two or more packets collide, each computer waits a random time and tries again later. In Figure 3.1, bus based broadcast network is shown.

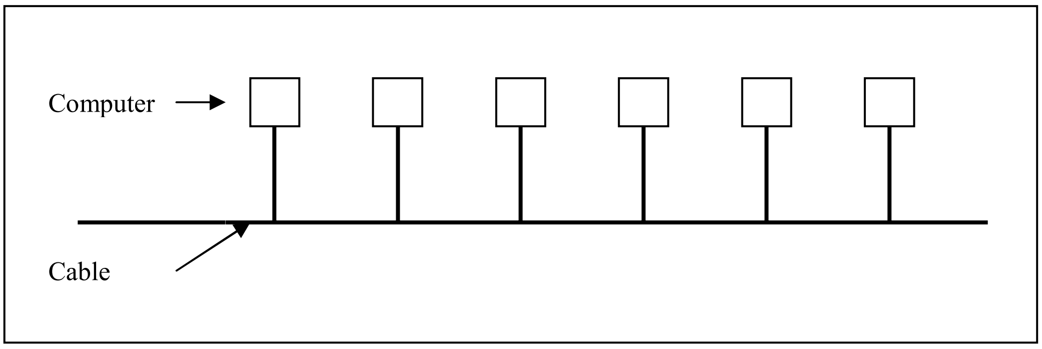

Figure 3.1 : Bus based broadcast network

The protocols used to determine who goes next on a multi-access channel belong to a sublayer of the data link layer called Medium Access Control (MAC) sublayer, [4]. Ethernet MAC uses the frame structure shown in Figure 3.2.

Preamble	Destination Address	Source address	Type	Data	Pad	Checksum
8 Bytes	6 Bytes	6 Bytes	2 Bytes	0-1500 Bytes	0-46 Bytes	4 Bytes

Figure 3.2: Ethernet Frame Format.

Each Frame starts with a preamble of 8 bytes, each containing the bit pattern of 10101010. The Manchester encoding of this pattern produces a 10-MHz square wave for 6.4 microseconds to allow receivers clock to synchronize with the sender’s. They are required to stay synchronized for the rest of the frame, using the Manchester encoding to keep track of the bit boundaries.

The frame contains two addresses, one for the destination and one for the source. It is up to the network layer to figure out how to locate the destination.

Next comes the “Type” field, which tells the receiver what to do with the frame. The type field specifies which processes give the frame to.

Next comes the “Data field”. Data length can extend up to 1500 bytes. In addition to the being a maximum frame length, there is also a minimum frame length. If data portion of a frame is less than 46 bytes, the “Pad” field is used to fill out the frame to the minimum size.

The final Ethernet field is the checksum. It is 4-byte in length. If some data bits are erroneously received (due to the noise on the cable) the check sum will almost certainly be wrong and the error will be detected. The checksum algorithm is a cyclic redundancy check. This algorithm performs only error detection not forward error correction.

3.2 Internet Protocol (IP)

Figure 3.3 shows the IP protocol header. The header has a 20 byte fix part and a variable length optional part, [4].

32 bits

Version	IHL	Type of service	▨	Total length			
Identification				▨	DF	MF	Fragment Offset
Time to live		Protocol		Header Checksum			
Source Address							
Destination Address							
Options (0 or more words)							

Figure 3.3: IP Header

The version field keeps track of which version of the protocol the datagram belongs to and it is 4 bits in length.

Since the header length is not constant, a field in the header, IHL, is provided to tell how long the header is, in 32 bit words. The minimum value of this 4-bit field is 5, which indicates there is no option present. The maximum value of this 4-bit field is 15, which limits the header to 60 bytes, and thus options field to 40 bytes.

The type of service field is intended to distinguish between different classes of service. It is used to indicate reliability and speed parameters. For digitized voice, fast delivery beats accurate delivery. For file transfer, error free transmission is more important than fast transmission. Originally, the 6-bits field contained, a 3-bits "Precedence" field and three flags *D, T* and *R.* The three flag bits allowed the host to specify what it cared most about the set {Delay, Throughput, and Reliability}. In theory, these fields allow routers to make choices between, for example, a satellite link with high throughput and high delay and a leased line with low throughput and low delay. In practice, current routers often ignore the Type of Service field altogether.

The Total length includes both header and data. The maximum length is 65535 bytes. At present, this upper limit is tolerable, but with future gigabit networks, larger datagrams may be needed.

The identification field is needed to allow the destination host to determine which datagram a newly arrived fragment belongs to. All the fragments of a datagram contain the same identification value.

DF stands for Don't Fragment. It is an order to the routers not to fragment the datagram because the destination is incapable of putting the pieces back together again. MF stands for More Fragments. All fragments except the last one have this bit set.

The fragment offset tells where in the current datagram this fragment belongs. All fragments except the last one in a datagram must be a multiple of 8 bytes.

The time to live field is a counter used to limit packet lifetimes. It is supposed to count time in seconds, allowing a maximum lifetime of 255 seconds. It must be decremented on each hop and is supposed to be decremented multiple times when queued for a long time in a router. In practice, it just counts hops. When it hits zero, the packet is discarded and a warning packet is sent back to the source host.

When the network has assembled a complete datagram, it needs to know what to do it. The protocol field tells it which transport process gives it to. The Header Checksum verifies the header only.

The Source address and Destination address indicate the network number and host number. The options field was designed to provide an escape to allow subsequent versions of the protocol to include information not present in the original design.

3.3. Real Time Protocol

Two standard transport protocols, TCP/IP and UDP are the most widely used protocols today, [4]. First, advantages and disadvantages of these protocols will be discussed and developed real time protocol will be explained.

All Internet Service Providers (ISP) support TCP/IP. Everyone with a home dial-up Internet account, home Digital Subscriber Line (DSL) account or home cable modem Internet account uses TCP/IP for communications with the Internet. TCP/IP refers to the format of data that is transmitted over the network and the rules in force for ensuring delivery at the desired location. TCP/IP is considered to be “reliable”. Reliable means that each individual packet that is sent over the network is verified at the receiver and acknowledged. If the data is larger than a single packet, it would be broken down into several individual packets and each would be transmitted separately. Packets are reassembled in the proper order at the destination prior to delivery to the client’s application. TCP/IP guarantees that packets will be reconstructed at the receiver in proper order. Reconstruction in the proper order is of vital importance to a voice signal. Out of order or lost packets will significantly degrade the quality of the transmitted voice. However, the processing overhead and delay for this guarantee will significantly increase latency in transmission and reconstruction of the voice signals.

UDP is the second-most widely used IP protocol in use. Unlike TCP/IP, UDP is unreliable. The UDP protocol does not contain the stringent requirement to acknowledge each individual packet. Packets are transmitted from the sender and essentially forgotten. While this reduces the overhead and delay in processing, packets can arrive out of order or be dropped from reception completely. Both of these protocols use an IP network for transmission. IP networks do not guarantee a specific path for delivery of packets between sender and receiver. Each packet may take a different network path, and can arrive to the destination out of order. For this reason, UDP is generally considered unsatisfactory for live voice. But UDP has advantages such as low protocol header overhead. Also, there is no retransmission mechanism in UDP. Retransmission mechanism in VoIP is not a practical option since the retransmitted packet will probably arrive too late to be useful.

The latest IP protocol developed specifically for streaming audio and video over the Internet is Real-Time Transfer Protocol. It is described in RFC 1889. RTP imposes packet sequencing and time stamping on a UDP data stream to ensure sequential packet reconstruction at the receiver while not imposing the high processing overhead of reliable transmission.

RTP protocol is adequate for Internet Radio, Internet Telephony (VoIP), video-on-demand, and other multimedia applications. In the scope of this thesis, a RTP like custom protocol is implemented which is only adequate for Internet Telephony applications. This protocol, as RTP, seats on top of UDP header

Custom RTP header is given in Figure 3.4. After UDP header of the IP/UDP packet, sequence number and timestamp information is added for voice packets into the data section of the UDP. Because RTP just uses normal UDP, its packets are not specially by the routers. In particular, there are no special guarantees about delivery and jitter. Also since retransmission mechanism introduces delay, there is no retransmission mechanism inserted into the RTP.

By using sequence numbers, packet sequencing at the receiver is enabled. Also, using sequence numbers allows the destination to determine if any packets are missing. If a packet is missing best action for the destination is receiver can determine lost packets and can take action for lost packets such as zero insertion.

For lost packets, interpolation or forward error correction algorithms can also be used. But these are not under the focus of this thesis.

Also, VoIP applications need timestamping. The idea here is to allow the source to associate a timestamp with the first sample in each packet. The timestamps are relative to the start of the stream. This mechanism allows the receiver to do a small amount of buffering and play each sample the right number of milliseconds after the start of the stream. By means of timestamping mechanism, effect of delay jitter is reduced on the played out speech.

An additional byte is used at the beginning of the RTP header for packet type identification and control purposes, which is referred as packet type identification character. Receiver of the packet starts some tasks according to the received packets packet type identification character; details are given in section 4.2.

Packet Type Identification Character (PTI)	Sender Timestamp (Ts)	Sequence Number (Sn)
1-Byte	4-Byte	2-Byte

Figure 3.4 : RTP Header

As mentioned before, RTP protocol is located over UDP. UDP transmits segments consisting of an 8-byte header followed by the payload. The header is shown in Figure 3.5. The two ports serve to identify the end points within the source and destination machines. When a UDP packet is received, its payload is handed to the process attached to the destination port. In fact, main value of the UDP over using just raw IP is the addition of the source and destination ports,[4]. Without the port fields, the transport layer would not know what to do with the packet.

Source Port	Destination Port
UDP Length	UDP Checksum

Figure 3.5: UDP Header

The source port is primarily needed when a reply must be sent back to the source.

The UDP "Length Field" includes the 8-byte header and the data. The UDP checksum is optional and stored as 0 if not needed.

It is probably worth mentioning explicitly some of the thing that UDP does not do. It does no flow control, error control or retransmission upon receipt of a bad segment. All of that is up to the user processes. What it does is to provide an interface to the IP protocol with the added feature of demultiplexing multiple processes using the ports.

After RTP packet is generated, it is embedded in UDP packet and UDP packet is embedded into the IP packet.. If the device is on a Ethernet network, as in our case, IP packets are then put in Ethernet frames for transmission. This packet nesting is shown in Figure 3.6.

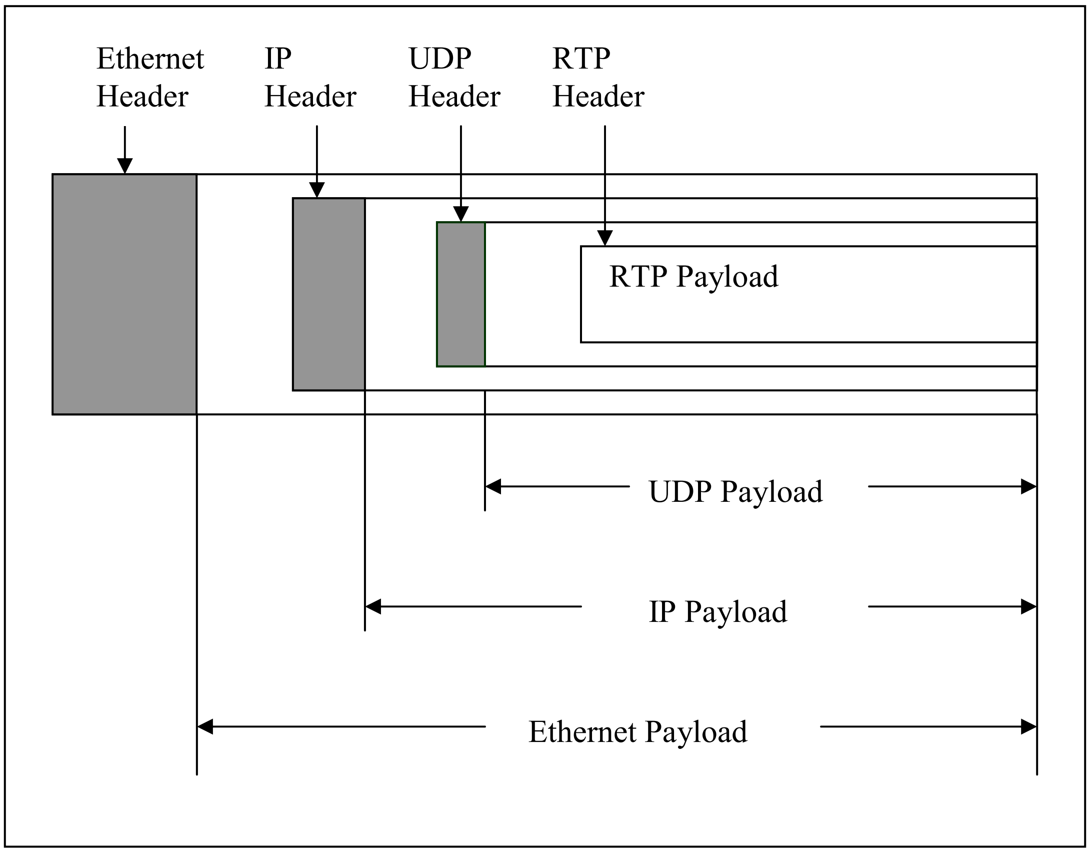

Figure 3.6 : Packet Nesting.

CHAPTER 4

SYSTEM ARCHITECTURE AND IMPLEMENTATION

In the framework of this thesis, an embedded Internet telephony device, providing full duplex voice communication over internet has been developed. Device performs connection establishment, voice packetization, voice activity detection, sending and receiving voice packets over a network using User Datagram Protocol (UDP), adaptive playout buffering of received packets to combat network delay jitter and converting these packets back to the voice. VoIP device is also able to measure network delay observed by the received packet, number of lost packets and round trip delay.

Developed board is placed in a box and UART connector, Ethernet connector, microphone input, speaker output, and Ethernet activity, collision and link leds, are placed on the box. VoIP device is seen in the Figure 4.2

In the following sections, developed board for the VoIP device is explained in detail, including both board hardware and software.

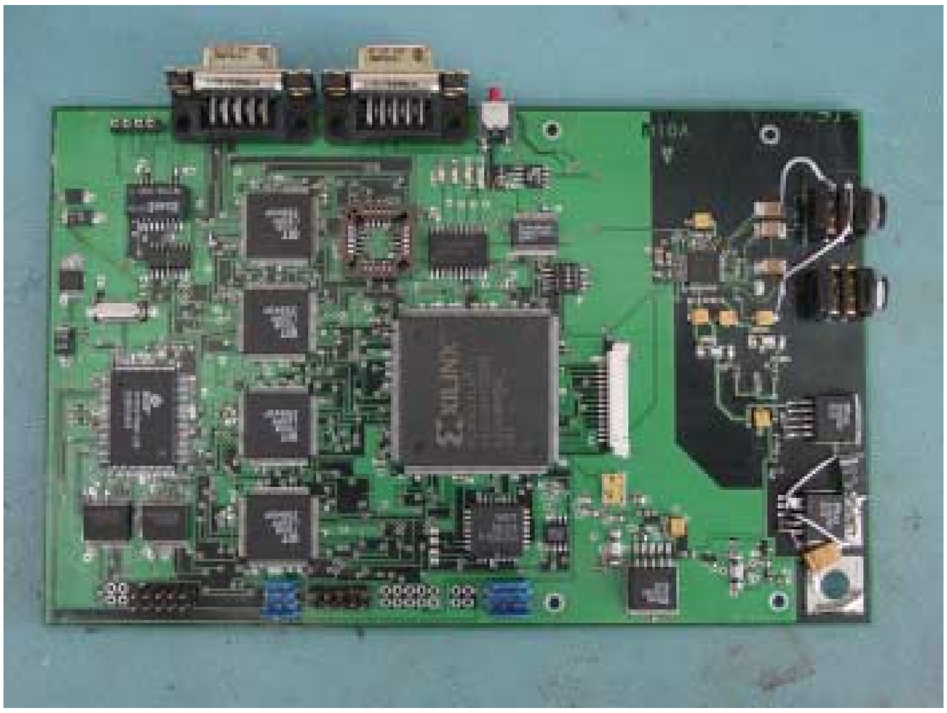

Figure 4.1: VoIP Board

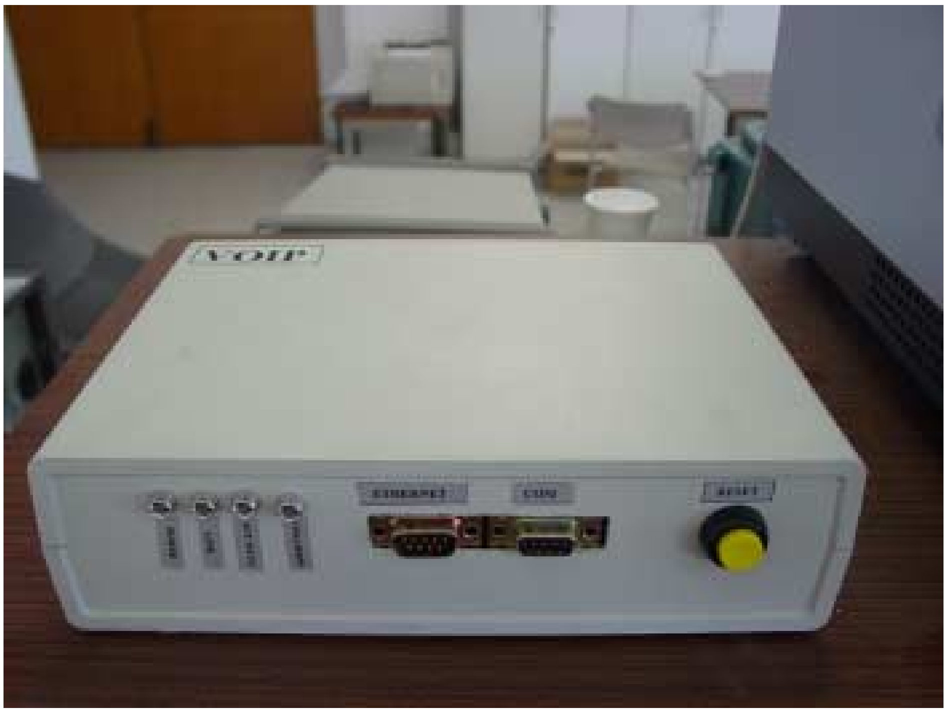

Figure 4.2: VoIP Device

4.1 Hardware

Analog voice signals are sampled and digitized using the Texas Instruments TLV320AIC22 voice codec, and then these samples are transferred to the Field Programmable Gate Array (FPGA) via the CODEC’ s serial interface. CODEC and FPGA use 24.576 MHz clock supplied by a crystal clock oscillator.

FPGA is the product of the Xilinx Inc. named XC4028. FPGA architecture is an array of logic cells that communicate with each other and pads via routing channels. Like semi-custom gate array, which consists of a sea of transistors, an FPGA consists of a sea of logic cells. In an FPGA, existing wire resources that run in horizontal and vertical columns (routing channels) are connected together via programmable elements, with logic cells and pads. Each logic cell consists of two programmable function generators, two flip flops and a multiplexer. Since it’s a reprogrammable device, there is flexibility in the development cycle and device upgrades. It is programming information is transferred on power up from a serial EPROM. ATMEL’s AT10V5 EPROM is used as configuration EPROM for the FPGA. This infrastructure gives enough freedom to be able to implement the necessary logic functions needed on the board.

FPGA design is performed using the “Very High speed integrated circuit hardware Description Language” (VHDL). VHDL is an IEEE standard for the description, modeling, and synthesis of circuits and systems. Because VHDL is a standard, VHDL design descriptions are device independent, allowing the designers easily benchmark design performance in multiple device architectures.

FPGA extracts ADC data from the codec serial interface. Formats the received data and transfer them to the microprocessor via the DPRAM interface. It also reads the received samples written to the DPRAM interface by the microprocessor and transfers them to the CODEC DAC section using the CODEC serial interface by performing necessary formatting.

Four 16Kx32 dualported RAM (DPRAM) are used for playout buffer memory requirements and to exchange voice samples between FPGA and microprocessor asynchronously

Microprocessor is the Ubicom's IP2022 processor, which is also called Internet processor. It is chosen for the supplied network stack, embedded Ethernet interface, in system programming and debugging capabilities. It uses 4.8 MHz crystal for its clock input and multiplies this frequency to reach the its operating speed of 120 MHz. This is a relatively high operating speed that minimizes the delay introduced on voice over IP communication by the application software. Microprocessor provides 10 MHz Ethernet interface of the board to send and receive voice packets. Ethernet interface is compliant to the IEEE 802.3 standard. Microprocessor also provides 57600 baud rate serial RS232 UART port to transmit diagnostics messages and for the reception of commands and device settings.

User interface of the device is provided by the serial interface of the microprocessor. Microprocessor communicates with the user interface program that runs on a Windows based PC, which is connected to the device via a serial port. By using user interface program, user can establish a connection with a host and can make device settings. User can send commands from user interface program to set device IP number, gateway number, subnet mask, to calculate clock offset, to enable voice activity detection and to start and stop round trip delay calculation. User interface program also logs the data, which is related to the received packets such as observed network delay and sequence number send.

Microprocessor performs Voice Activity Detection (VAD) on voice samples, which are supplied by the FPGA using DPRAM interface. Packets the voice samples and sends them to the receiving host using Ethernet interface. A playout-buffering algorithm runs on microprocessor to assign playout time to the received packets. Received voice packets are transferred from the network to the FPGA using

DPRAM interface to be played out by the DAC according to the assigned playout time.

In Figure 4.3 block diagram of the device is shown. In the following subsections board hardware, FPGA and microprocessor software and user interface program are explained in detail.

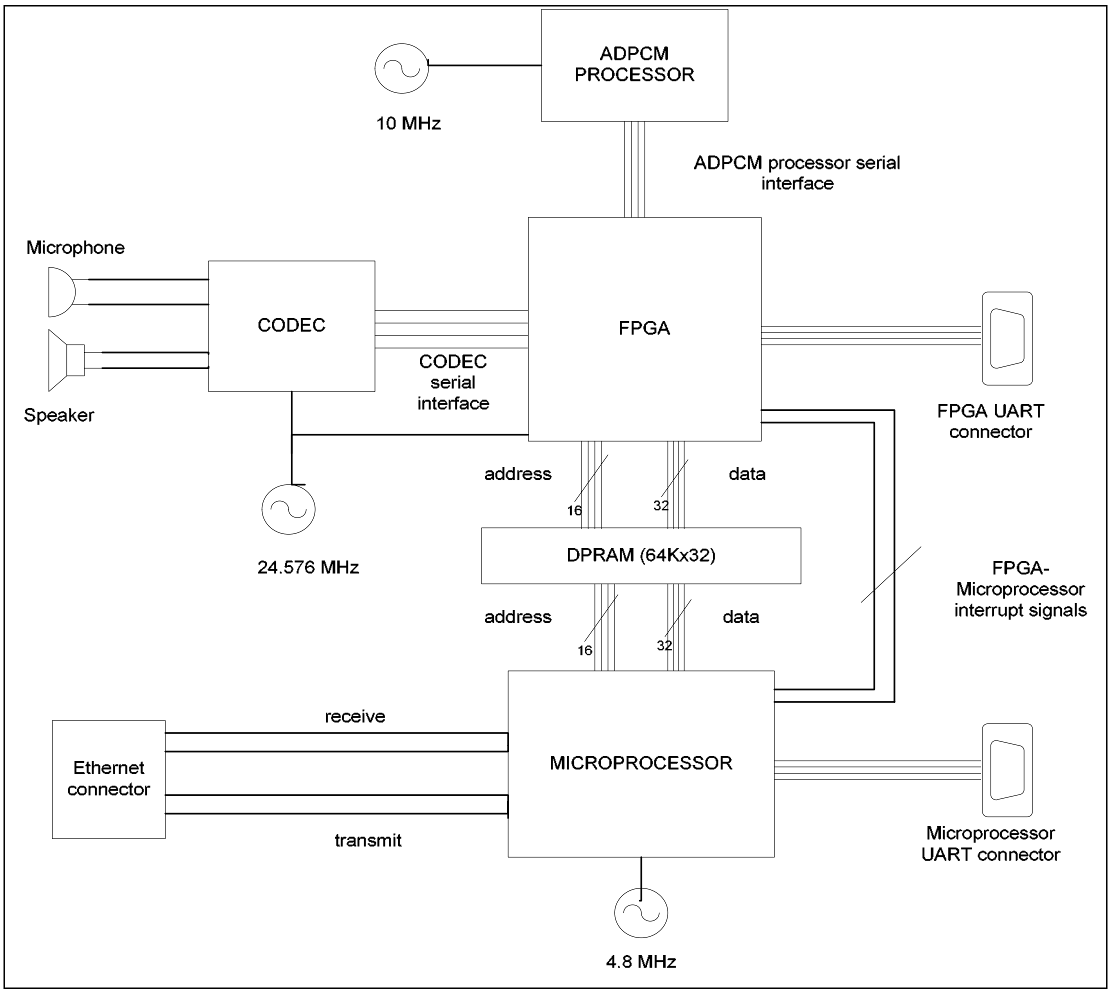

Figure 4.3: Block diagram of the VOIP board

4.1.1 Voice Digitization

For voice digitization Texas Instruments TLV320AIC22 voice codec is used. Device performs both of the analog to digital and digital to analog conversion of voice. Sampled values of the input voice signal are passed to the FPGA and voice samples to be converted to the analog signal are received from the FPGA via a CODEC serial interface.

Functional block diagram of the codec is given in Figure 4.4.

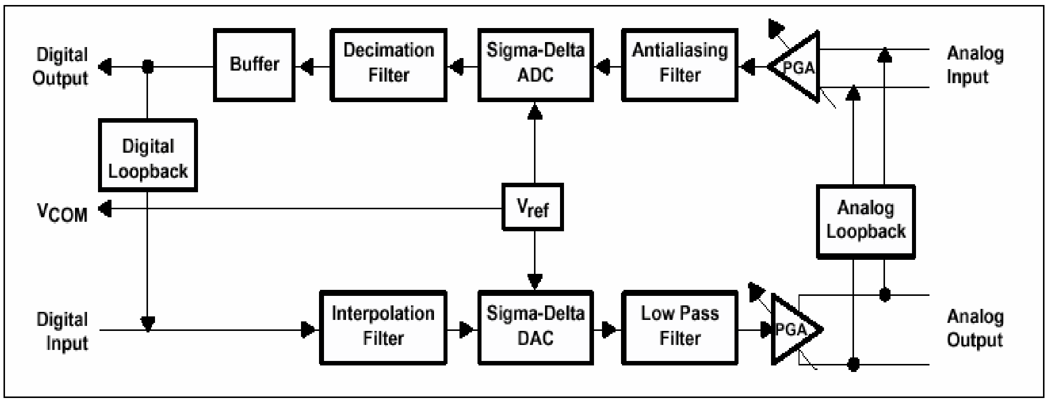

Figure 4.4: Functional block diagram of the CODEC

The ADC channel consists of a programmable gain amplifier (PGA), antialiasing filter with a 3.6 kHz bandwidth for a sampling rate of 8 kHz, sigma delta analog to digital converter and decimation filter. The ADC is an over sampling sigma-delta modulator. The ADC provides high resolution and low-noise performance using over sampling techniques and the noise shaping advantages of sigma-delta modulators.

The analog input signals are amplified and filtered by on-chip buffers and an antialiasing filter before being applied to ADC input. The ADC converts the analog voice signal into discrete output digital words in 2s-complement format, corresponding to the analog signal value at the sampling time.

The decimation filter reduces the digital data rate to the sampling rate. This is accomplished by decimating with a ratio equal to the over sampling ratio. The output of this filter is a 16-bit 2s-complement data word clocking at the selected sample rate. Output samples are then compressed to the 8-bit μ-law PCM format.

8 bit μ-law PCM digital words, representing sampled values of the analog input signal, are sent to the FPGA via the serial port interface. The ADC and DAC conversions are synchronous.

The DAC channel consists of an interpolation filter, a sigma-delta DAC, low-pass filter, and a programmable gain amplifier. The DAC is an over sampling sigma-delta modulator. The DAC performs high-resolution, low-noise, digital-to-analog conversion using over sampling sigma-delta techniques.

The DAC receives 8 bit μ-law PCM words from the FPGA via the serial port interface.

The data is converted to an analog voltage by the sigma-delta DAC comprised of a digital interpolation filter and a digital modulator. The interpolation filter resample the digital data at a rate of 2 times the incoming sample rate where 2 is the over sampling ratio. The high-speed data output from this filter is then applied to the sigma-delta DAC.

The DAC output is then passed to an internal, low-pass filter to complete the signal reconstruction resulting in an analog signal. This analog signal is then buffered and amplified by differential output driver capable of driving 150 ohm microphone load.

4.1.2 CODEC- FPGA Serial Interface

As mentioned before data exchange between COEDC and FPGA is performed via a serial codec interface. Both of the FPGA and CODEC runs with 24.576 MHz master clock (MCLK). For serial data exchange between them, codec behaves as master and provides 2.048 MHz bit clock (BCLK) and frame synchronization pulse (FSYNC) for every 256 BCLK cycles. Two other lines Dout, which is used to transfer ADC data to the FPGA and Din, which is used to transfer DAC data and codec programming information from the FPGA are used. Timing diagram of the codec serial interface is given in Figure 4.5

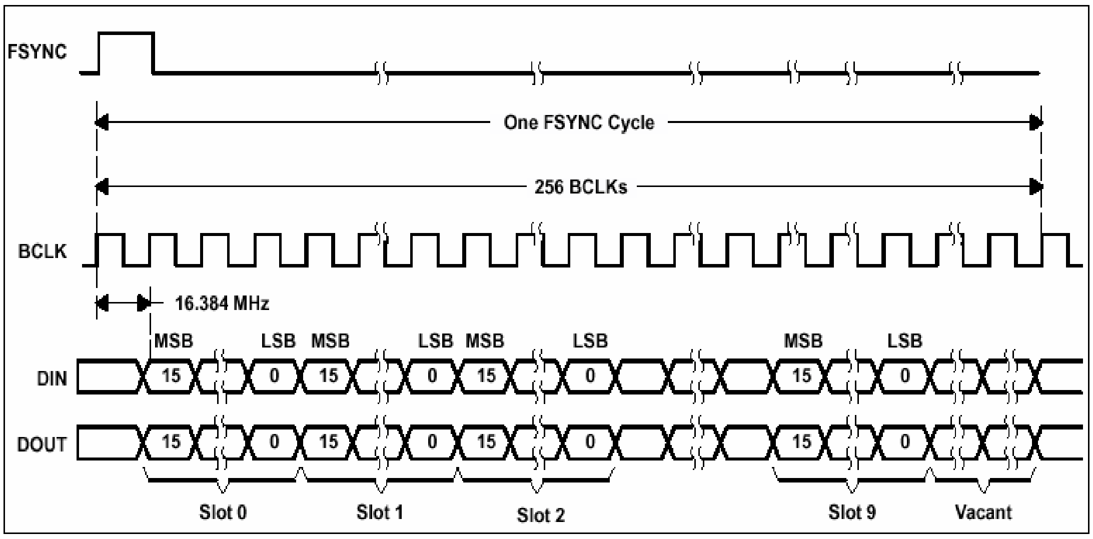

Figure 4.5: Timing diagram of the Codec serial interface

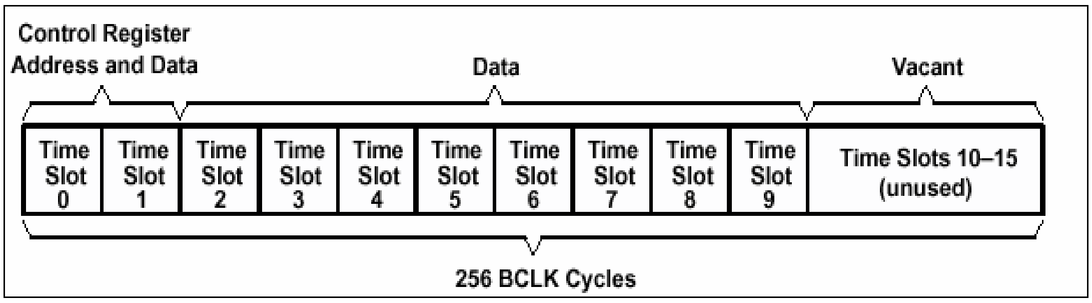

Figure 4.6 : CODEC serial interface frame format

Data is received and transmitted in frames consisting of 256 BCLKs, which is sixteen, 16-bit time slots. Each frame is subdivided into time slots consisting of 16 BCLKs per time slot. In each frame, two time slots are reserved for control register information and eight time slots are reserved for codec data. The remaining six time slots are unused. A pulse on the FSYNC pin indicates the beginning of a frame.

FPGA uses rime slot 2 to exchange ADC DAC data. In time slots 0 and 1 FPGA sends programming information to the codec. FPGA selects microphone input pins of the CODEC as ADC input and headphone output pins of the CODEC as DAC output. PGA amplifier gain is set to 12 dB, preamplifier gain is set to 23 dB for both the ADC and DAC networks. Echo gain for the headphone output is set to -12 dB to cancel the echo coupled to the DAC output from the ADC input. These values are chosen experimentally to give a satisfactory audio level.

4.1.3 ADPCM Compression- Expansion.

APCM compression option is added to the device to allow transmitted bit rate reduction. DS2165 ADPCM Processor Chip is used for ADPCM compression and expansion. DS2165 ADPCM Processor Chip is a dedicated Digital Signal Processing (DSP) chip that has been optimized to perform Adaptive Differential Pulse Code Modulation (ADPCM) speech compression. The chip can be programmed to compress (expand) 64 kbps voice data down to (up from) either 32 kbps, 24 kbps, or 16 kbps. The compression to 32 kbps follows the algorithm specified by CCITT Recommendation G.721 (July 1986) and ANSI document T1.301 (April 1987). The compression to 24 kbps follows ANSI document T1.303. The compression to 16 kbps follows a proprietary algorithm developed by Dallas Semiconductor.

The DS2165 contains three major functional blocks: a high performance (10 MIPS) DSP engine, two independent PCM interfaces (X and Y) which connect directly to serial Time Division Multiplexed (TDM) backplanes, and a serial port that can configure the device on-the-fly via an external controller. A 10 MHz master clock is required by the DSP engine. Each channel on the device samples the serial input PCM or ADPCM bit stream during a user-programmed input time slot, processes the data and outputs the result during a user-programmed output time slot.

Onboard counters establish when PCM and ADPCM I/O occur. The counters are programmed via the time slot registers. Time slot size (number of bits wide) is determined by the working state of the device; compression or expansion.

For example, if the X channel is set to compress then the input port (XIN) is set up for 32 8-bit time slots and the output port (XOUT) is set up for 64 4-bit time slots.

4.1.4 FPGA-ADPCM Processor Interface

Since the organization of the input and output time slots on the DS2165 does not depend on the algorithm selected, it always assumes that PCM input and output will be in 8-bit bytes and that ADPCM input and output will be in 4-bit bytes. Figure 4.7 demonstrates how the DS2165 handles the I/O for the three different algorithms. In the figure, it is assumed that channel X is in the compression mode and channel Y is in the expansion mode Also, it is assumed that both the input and output time slots for both channels are set to 0.

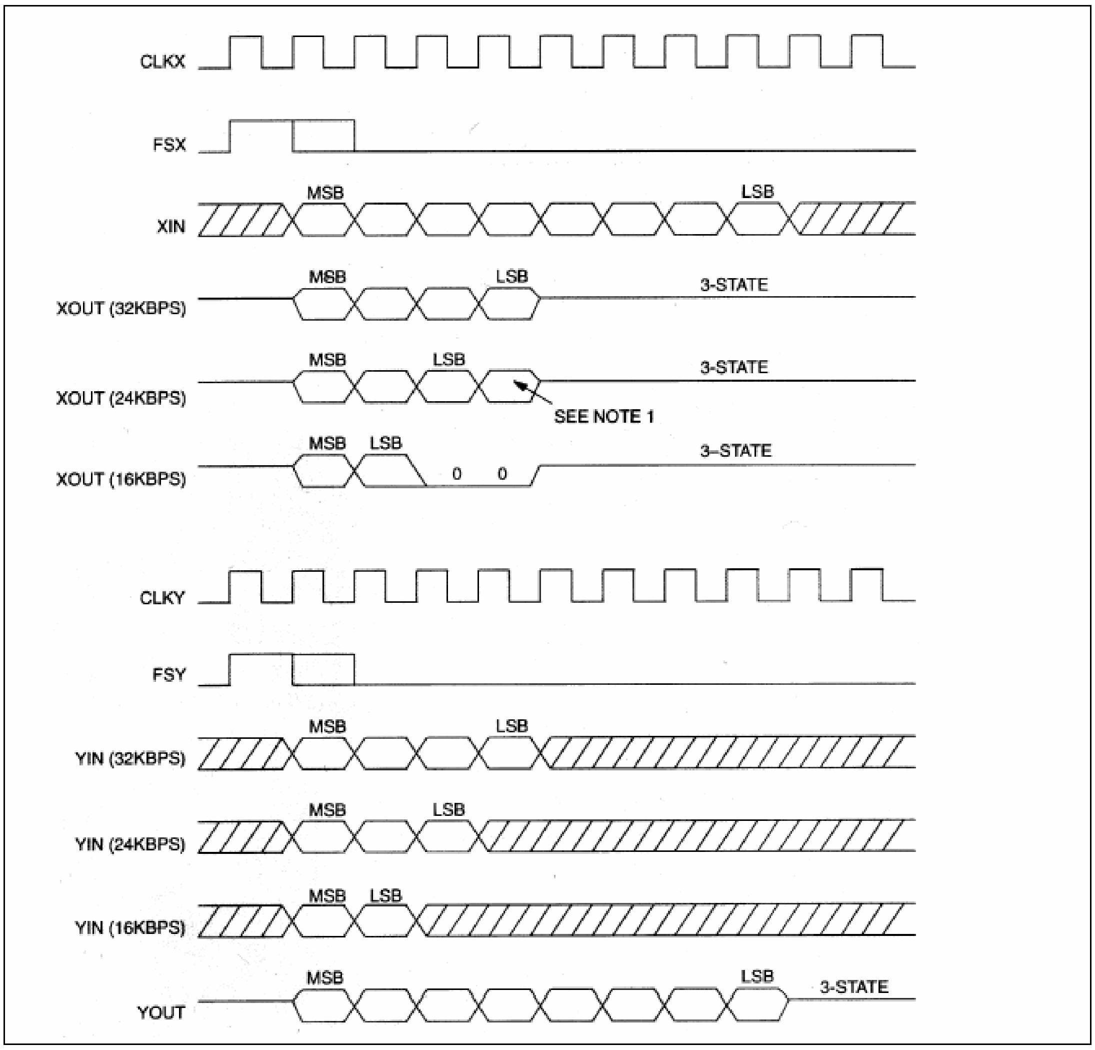

Figure 4.7: FPGA-ADPCM Processor Interface

In designed board, X channel is used for compression ad Y channel is used for expansion. Algorithm for compression to the 16 Kbps is used. When FPGA extracts PCM voice sample from codec serial interface, it sends the PCM voice sample to the ADPCM processor for compression by implementing the ADPCM interface which is shown in Figure 4.7. At the same time, it extracts ADPCM compressed samples from the Xout pin of the processor. Extracted samples are written to the DPRAM ADC data section by completing 2 bits APCM samples to the 8 bits by zero padding. This operation allows microprocessor software work with both ADPCM and PCM data. Microprocessor extracts compressed sample when in ADPCM mode of a operation by masking the zero padded bits.

For expansion, FPGA reads the compressed samples from the DPRAM interface, which are written by microprocessor, and then it sends the compressed sample to the ADPCM processor using the Y channel. Expanded samples are extracted from the Yout pin of the processor and send to the CODEC for playout.

These operations are performed when device is in ADPCM mode of a operation. Otherwise they are bypassed and PCM voice samples are transferred to the microprocessor. Microprocessor informs FPGA for mode of a operation by a single line connection between FPGA and microprocessor. Logic high value on this line corresponds to the PCM mode and logic low value corresponds to the ADPCM mode.

4.1.5 Microprocessor-FPGA interface:

FPGA runs at 24.576 MHz clock rate and microprocessor runs at 120 MHz clock rate. Their clocks are asynchronous. Direct connection of FPGA and microprocessor can cause unestimated communication problems between them because of asyncronicity. Therefore, their communication is performed over a asynchronous 4 16Kx32 Dual Ported RAM (DPRAM). These four DPRAMs are connected as a single 64Kx32 DPRAM.

DPRAM has two ports, which are completely independent of each other. Each of the ports has address bus width of 16 bits and data bus width of 32 bits. All memory locations are available to the both of the ports. Microprocessor uses one of the ports to read/write data to the DPRAM and FPGA uses the other port. Therefore, there is no direct connection between them and data exchange is performed by memory read write operations.

ADC data is written by the FPGA to the DPRAM ADC data section and it is read by the microprocessor. DAC data, received from the network is written to the DAC data section and it is read by the FPGA to play the received audio. Two interrupt lines exist between FPGA and microprocessor to inform each other that the data is ready at the DPRAM.

Because of the low available I/O pin count of the Microprocessor, address and data lines of DPRAM interface uses same set of 16 pins. An additional line is used to demultiplex the address and data lines before connecting them to the DPRAM. Because of data width of the DPRAM is 32 bit, data is written or read from the DPRAM by the Microprocessor in two cycles by enabling one half of the DPRAM at a time. Block diagram of Microprocessor DPRAM interface is given in Figure 4.8. FPGA performs read and write operations in one cycle by the logic circuit designed inside the FPGA.

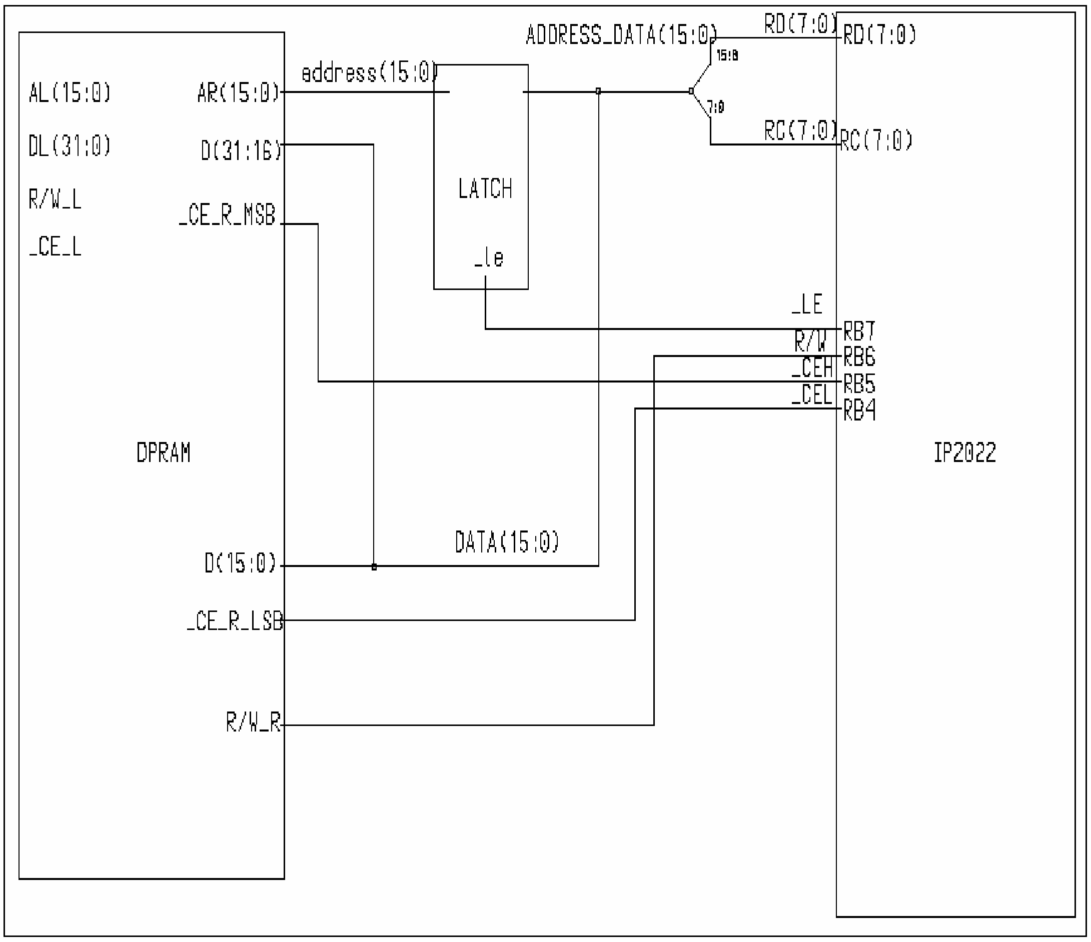

Figure 4.8: DPRAM- Microprocessor Interface

As mentioned, DPRAM data width is 32 bits. Therefore four voice samples are written per address location of the DPRAM. These samples can be PCM samples or ADPCM samples depending on mode of a operation. These packed samples will be called DPRAM word. Data format is shown in Table 4.1.

Table 4.1: DPRAM data format

Sample N+3	Sample N+2	Sample N+1	Sample N

Since the sampling frequency is 8 KHz, every word in the DPRAM corresponds to 0.5 ms of voice. Voice packets that are used for voice over IP communication carries 16 ms of voice, which corresponds to 128 samples. When 128 samples are written to the 32 sequential locations of DPRAM from the start of the ADC data section 1, FPGA interrupts the microprocessor to inform that ADC data is ready in ADC data section1 for further processing. When microprocessor reads samples from ADC section 1, FPGA writes new samples to the ADC DATA section 2 and interrupts microprocessor again. An additional line is used by the FPGA to inform the microprocessor, which ADC data section contains new samples This process continues sequentially. DPRAM sections are given in Table 4.2.

TABLE 4.2: DPRAM Sections

ADC Data Section 1 Start	0x0000
ADC Data Section 1 End	0x001F
ADC Data Section 2 Start	0x0020
ADC Data Section End	0x003F
DAC DATA Section Start	0x0080
DAC DATA Section End	0x3F00

DPRAM DAC section is used as a circular buffer by FPGA and microprocessor. FPGA reads one DPRAM word every 0.5 ms and increments DPRAM address by one by starting from the DAC DATA section start location, once it is informed that connection is established. When FPGA reaches to the end of the DAC DATA section it changes DPRAM address to the DAC DATA section start address. Microprocessor writes received packets to the DPRAM location, where packet playout time, calculated by the playout-buffering algorithm, refers.

4.2 Microprocessor Software

Microprocessor manages the board. It supplies board's Ethernet interface and RS232 UART port. Microprocessor software development environment provides an operating system, UART and Ethernet drivers and network protocol software stack.

Operating system provides necessary parellization between the tasks by using timer based interrupt subroutine. By using external interrupt line that is connected to the FPGA, FPGA-microprocessor communication is carried out over DPRAM's.

Microprocessor software performs three main functions.

i. Control Functions.
ii. Sender Functions.
iii. Receiver Funcitons.

4.2.1 Control Functions.

Board can be controlled using the UART port of the microprocessor by the user. Microprocessor performs assigned tasks when it receives commands from the UART interface.

Also, some sort of control information should be carried between the communicating two host VOIP devices. These control information are inserted into the transmitted UDP packet's data section and named packet type identifier. When VOIP device receives a UDP packet, it takes some actions according to the received packet's packet type identifier.

When microprocessor receives a packet from a UART or ETHERNET interface, interrupt subroutine of microprocessor invokes processes for each of them, named *uart_data_receive* and *ethernet_data_receive* respectively.

When commands are received from UART interface, *uart_data_receive* process invokes the processes that run the tasks indicated by the received command. These commands are given in Table 4.3 and tasks related to the each command are explained in the following subsections.

TABLE 4.3: UART Commands

COMMAND	PAREMETER	FUNCTION
‘i’	4 byte IP Number	Set IP Number
‘s’	4 byte Subnet Mask	Set Subnet Mask
‘g’	4 byte Gateway Number	Set Gateway Number
‘r’	4 byte IP Number of the Host	Calculate Clock Offset
‘c’	4 byte IP Number of the Host	Connect
‘q’	-	Disconnect
‘e’	-	Enable VAD
‘d’	-	Disable VAD
‘t’	4 byte IP Number of the Host	Start RTT measurement
‘f’	-	Stop RTT measurement
‘p’	2 byte RTT Packet Length	Set RTT Packet Length
‘l’	2 byte RTT measurement period	Set RTT measurement period
‘a’	-	Set ADPCM mode
‘b’	-	Set PCM mode

All UDP packets, constructed by the microprocessor, include a one-byte packet type indicator character at the first byte of the UDP data section. When UDP packets are received, *ethernet_data_receive* process first checks the packet type indicator of the received packet and executes the related tasks. Packet type indicators are given in Table 4.4. Tasks related to the packet type indicators are explained in the following subsections.

TABLE 4.4: Packet Type Indicator Characters

UDP PACKET TYPE IDENTIFIER	UDP PACKET TYPE
‘c’	Connection Request Packet for PCM mode
‘a’	Connection Request Packet for ADPCM mode
‘q’	Disconnect Packet
‘f’	Clock Offset Calculation Request Packet
‘p’	Clock Offset Calculation Reply Packet
‘v’	Voice Packet
‘s’	Voice Packet. (Start of silence)
‘r’	RTT measurement initiator packet
‘a’	RTT measurement acknowledge packet

Also, software drivers are written for microprocessor to be able to read and write to the external DPRAM memory. Microprocessor reaches DPRAM using three of its ports as seen in Figure 4.5.

4.2.1.1 Network Settings

Boards network settings should be made when it is inserted in a new Ethernet network. It’s IP number, subnet mask and gateway numbers should be chosen that can be used in that network and they should be assigned to the board using UART interface. For that purpose, three commands are assigned. These are named SET IP NUMBER, SET SUBNET MASK and SET GATEWAY NUMBER

VOIP device’s network numbers are stored in the microprocessors internal FLASH memory. FLASH memories are nonvolatile memories. Therefore board stores these numbers when its power is off. When boards power is on microprocessor reads these numbers from its FLASH memory to its DATA memory. This is done because that microprocessor reaches its DATA memory faster than FLASH memory. DATA memory can be accessed at 120 MHz and FLASH memory at 30 MHz.

One byte Network number setting commands are followed by four byte network numbers that will be written to the internal FLASH memory as new network number. When microprocessor receives network number setting commands it waits for four following bytes. After it receives these four bytes, it changes the related network number at the FLASH memory.

FLASH memory is organized as 4 blocks of 512 bytes. If one wants to change a line in a block of flash memory, all the block contents should be erased first before writing to the block. Network numbers are located at third block. Therefore when microprocessor changes one of the network numbers, it transfers all three network numbers to its data memory. Changes the network number according to the received command in the data memory. Erases third block of flash memory and then transfers all three network numbers from data memory to the flash memory including the one changed.

4.2.1.2 Clock Offset Calculation

When the packet arrives at the receiver host, the delay is calculated using the receiver’s clock. In this method, however, time clocks of the sender and receiver should be synchronized in order to measure accurate delays. Unless the two clocks are not synchronized, different two clocks may cause relative offset and skew as illustrated in Figure 4.9. The relative offset of the two clocks is caused when the two clocks have different time.

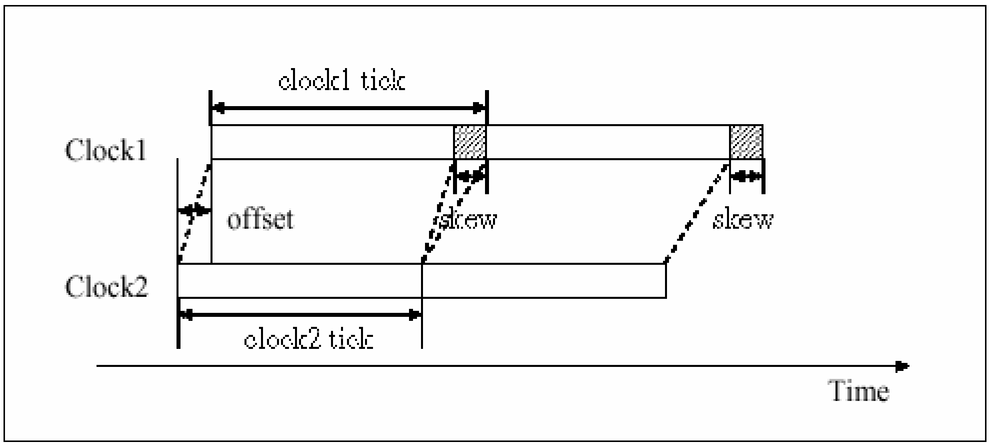

Figure 4.9 : Clock offset and skew between two clocks

Before connecting to a host, synchronization should be established between them. Therefore a command is assigned for clock offset calculation. Clock offset is calculated using the method given in [5].

When clock offset calculation command is received from UART interface, microprocessor invokes a process. This process constructs a UDP packet for clock offset calculation request and sends it to the host. First byte of the UDP packet includes packet type indicator character ‘f’ that indicates this is a clock offset calculation request packet. Following four bytes includes the timer value of the microprocessor at the UDP packet’s construction instant.

When host receives clock offset calculation request packet, it immediately reads the value of its timer. Then it adds this value to the end of the received UDP packet’s data section. Changes packet type indicator character to ‘p’ that indicates this is a clock offset calculation reply packet. Finally it reads its timer value again, that shows the send instant. Writes this value to the end of the constructed packet. Sends this packet back to the requesting communication party.

When clock offset calculation requester receives clock offset calculation reply packet, it reads its timer value immediately and stores it as receive time. Then calculates the clock offset value.

UDP packet exchange for clock offset calculation between clients A and B is seen in Figure 4.10. Client A request clock offset calculation and client B replies it. Data sections of the UDP packets that are used for clock offset calculation between clients A and B is seen Figure 4.11.

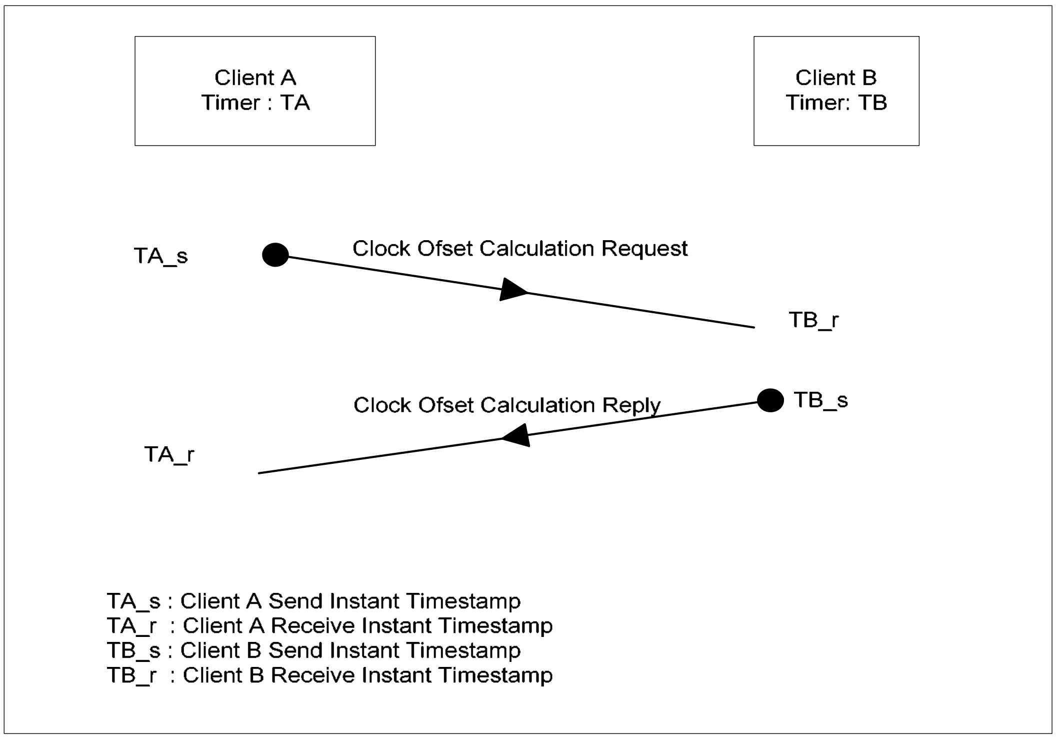
Client A
Timer : TA
Client B
Timer: TB
TA_s
Clock Ofset Calculation Request
TB_r
Clock Ofset Calculation Reply
TB_s
TA_r
TA_s : Client A Send Instant Timestamp
TA_r : Client A Receive Instant Timestamp
TB_s : Client B Send Instant Timestamp
TB_r : Client B Receive Instant Timestamp

Figure 4.10: UDP packet exchange for clock offset calculation between clients A and B

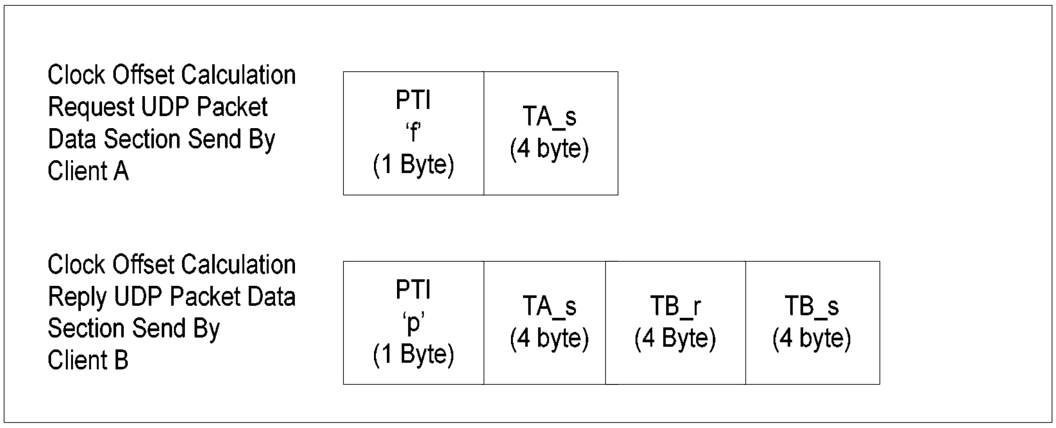
Clock Offset Calculation Request UDP Packet Data Section Send By Client A
PTI 'f' (1 Byte)
TA_s (4 byte)
Clock Offset Calculation Reply UDP Packet Data Section Send By Client B
PTI 'p' (1 Byte)
TA_s (4 byte)
TB_r (4 Byte)
TB_s (4 byte)

Figure 4.11: UDP Packet's Used for Clock Offset Calculation Between clients A and B

When client A receives clock offset calculation reply packet from client B, it immediately records the timer value, which is the client A receive instant timestamp *TA_r*.

It is assumed that, network delay observed by the request and reply packets are same. This assumption makes the calculation of clock offset possible. Let's says both of the packets observe network delay of *nd* milliseconds and timer value of the client B is higher than the timer value of the client A by *ΔC* ms offset. Then following equations can be written for receive times.

$$TB\_r = TA\_s + \Delta\mathrm{C} + \mathrm{nd} \qquad (4.1)$$

$$TA\_r = TB\_s - \Delta\mathrm{C} + \mathrm{nd} \qquad (4.2)$$

By manipulating the equations 4.1 and 4.2 equation 4.3 is obtained for clock offset calculation.

$$\Delta C = \frac{(TB\_s + TB\_r) - (TA\_s + TA\_r)}{2} \qquad (4.3)$$

Microprocessor calculates the clock offset using equation 4.3 after the clock offset calculation reply packet is received as given in [5].

4.2.1.3 Connection Establishment

After synchronization is established between the communication clients, connection can be established. Connect command; received from UART interface starts the connection establishment tasks.

Connect command is a five byte command, ‘c’ or ‘a’ character, depends on the mode of a operation PCM or ADPCM, followed by four byte IP number of the host to be connected, . When microprocessor receives ‘c’ or ‘a’ character from its UART port, it waits for 4-byte IP address of host. After receiving IP number, it turns its UART interface to the new command wait state and calls its connection establishment task.

Connection establishment task, constructs a UDP packet. This UDP packet is three bytes long. First byte is packet type identification character ‘c’ that indicates this is a connection request packet. Following two bytes are clock offset value, calculated before connection establishment. Microprocessor changes the sign of the clock offset value before writing it to the UDP packet since the clock offset for the host to be connected is opposite of the clock offset calculated.

When host receives the connection request packet it reads the clock offset value and records it for delay calculation. Converts microprocessors state to the connection-established state. Then, microprocessor sends a 1-byte long UDP packet that only consists of packet type identification character ‘o’ that indicates this is a connection acceptance packet. When the connection requester receives the connection acceptance packet it turns its microprocessor state to the connection-established state too. In this state, microprocessor does not respond to the connection request packets.

Connection is terminated by a 1-byte disconnect command received form UART interface. ‘q’ character received from UART interface indicates connection should be terminated.

When microprocessor receives ‘q’ character from UART interface, it changes its state to the no connection state and stops sending voice packets to the host. It constructs a UDP packet that is one byte long, containing packet type identification character ‘q’ that indicates this is a connection termination packet. When host receives connection termination packet, it changes its microprocessor state to the no connection state and stops sending voice packets to the host.

4.2.1.4 RTT Measurement

Round Trip time measurement facility is added to the VOIP device to be able to extract the characteristics of the network that device is inserted. RTT measurements give more accurate delay measurement results than one-way delay measurement results, because of there are no synchronization problem between the hosts; because all calculations are made using the RTT measurement initiator’s clock..

Also, data length of the RTT packets and time period between departure times of the two successive RTT packets can be adjusted using the serial control interface. This gives chance to extract the network characteristics with different settings of this variables. By default, RTT packet length is set to 128 bytes and RTT measurement period is set to 20 ms.

RTT packet length is set with 3-byte command received from UART interface. This command consists, ‘p’ character followed by 2-byte number, indicating RTT packet length in bytes. When microprocessor receives this command from its UART interface, it changes the value of the variable indicating RTT packet length to the received value.

In the same way, RTT measurement period is set by the 3-byte command received from UART interface. This command consists ‘l’ character followed by 2-byte number, indicating RTT measurement period in milliseconds. When microprocessor receives this command from its UART interface, it changes the value of the variable indicating RTT packet length to the received value.

After packet length and period settings are done, RTT measurement can be started with RTT measurement start command send through UART interface of the microprocessor. RTT measurement start command is a 5-byte command,‘t’ character followed by a 4-byte IP number of the host, which RTT measurement will be made with. When microprocessor receives RTT measurement start command, it constructs a UDP packet that will be send to the host and that packet will be acknowledged by the host. This packet consists packet type identification character ‘r’, indicates that this is rtt measurement initiator packet, at the first byte of the data section of the UART packet. Packet type identifier character is followed by current value of the microprocessor timer, which is 4-byte in length, and 2-byte sequence number. After that, toggling binary 1’s and 0’s are inserted to the packet’s data section until packet size reaches the RTT packet length. As soon as packet is constructed, it is send to the host that measurement will be made with. This process of packet construction and transmission to the host repeats every RTT measurement period elapses until the RTT measurement is stopped. Sequence number is incremented by one for every transmitted packet.

When host receives RTT measurement initiator packet, it immediately changes packet type identification character of the received character from‘t’ to ‘a’, that indicates this is a RTT measurement acknowledge packet, and transmits packet back to the RTT measurement initiator. When RTT measurement imitator receives acknowledge packet, it immediately records current value of the microprocessor timer. Then, it reads sequence number and departure time of the packet from the beginning of the packets data section, calculates RTT by subtracting the departure time from the recorded value of the timer when acknowledge packet is received. Calculated RTT and corresponding sequence number is send from UART interface to be logged.

RTT measurement stops with RTT stop command received from UART interface. This command is one byte command, which is the ‘f’ character.

4.2.2 Sender and Receiver Functions

Sender and receiver functions cover, sending and receiving voice UDP packets through Ethernet interface to satisfy satisfactory voice communication over IP. Before sending voice packets, voice activity detection is performed. At the receiving side, adaptive or fixed playout buffering is performed on received packets before playout.

Voice samples are transmitted to the receiver using developed RTP that is explained Chapter 3. A voice UDP packet consists of packet type identification character ‘v’, 2-byte sequence number of voice UDP packets, 4 byte timestamp information that is used to show the packets generation instant and 128 8-bit μ-law compressed voice samples or 128 2-bit APCM compressed voice sample depending on a mode of a operation. Since the sampling frequency is 8 kHz, 128 voice samples carry voice information that is 16 ms of duration. A voice UDP packet data section is total of 135 bytes in length and arranged as in Figure 4.12 for PCM mode of a operation and 39 bytes for ADPCM operation and arranged as in Figure 4.13.

← 135-byte →

RTP Header 7-byte	Sample(n) 1-byte	Sample(n+1) 1-byte	………	Sample(n+126) 1-byte	Sample(n+127) 1-byte

Figure 4.12: Voice UDP Packet Data section For PCM mode

← 39-byte →

RTP Header 7-byte	Sample(n) 2-bits	Sample(n+1) 2-bits	………	Sample(n+126) 2-bits	Sample(n+127) 2-bits

Figure 4.13: Voice UDP Packet Data section for ADPCM mode

4.2.2.1 Sender Functions

These part of the microprocessor software is responsible for reading voice samples from DPRAM, generating voice UDP packets and sending these packets to the host after connection is established.

Every 16 ms, FPGA writes 128 voice samples to the one of the DPRAM ADC DATA sections and interrupts microprocessor to inform that ADC data is ready. Microprocessor interrupt service routine invokes *send_adc_data* task, if connection is established. Pseudo code for the *send_adc_data* task is given in Appendix B.

S*end_adc_data* task, when invoked starts forming voice UDP packet. First, timer value of the microprocessor is read and recorded as a voice packet generation timestamp. Then packet sequence number is incremented. At the receiver side timestamps are used for delay calculation and the sequence numbers are used to arrange the received packets in the correct generation order.

After that, microprocessor allocates a buffer inside its internal memory. Voice UDP packet data section will be formed in this buffer. Packet type indication character ’v’, which indicates this is a voice packet, is written to the start of the buffer. Then packet generation timestamp that is four-byte and two-byte packet sequence number is written to the buffer. These 7 byte data forms the RTP header of the voice packet as explained in Chapter 3, Real Time Protocol section.

After then, ADC data section of the DPRAM, that contains new samples, should be determined. This task is accomplished by reading the logic level of the corresponding pin of the microprocessor, which is driven by the FPGA. FPGA drives this pin with logic level of low to indicate that first ADC data section contains new samples and with logic level of high to indicate that second ADC data section contains new samples. After the ADC data section that contains new samples is determined, voice samples are read from the determined DPRAM ADC data section that contains 128 samples or 32 DPRAM words which are 4 bytes long as indicated in section 4.1.3.

Voice activity detection is performed on voice samples to decide start, continue or stop voice UDP packet transmission. Microprocessor compares every sample read from DPRAM with a voice activity detection threshold and writes them to the internal buffer, which is allocated to form the voice UDP packet data section, sequentially. Voice activity detection threshold is selected experimentally. Samples that are over the threshold are called active samples. For each active sample, a counter called active sample count is incremented by one. After all of the 128 samples are read, the number of samples that are over the voice activity threshold is checked using the value of the active sample count. If number of active samples is greater than the half of the number of samples, all of the 128 samples are treated as containing voice activity. Otherwise all of the 128 samples are treated as not containing voice activity or silent.

After all of the samples are written to the internal buffer, voice UDP packet data section is formed. This packet is send over network using UDP protocol to the receiving host if silence period is not determined. If silence period is determined, voice packet transmission is stopped until the end of the silence period is determined.

Start of silence period is decided when sequential 3 voice packets read from DPRAM is silent. First two packets at the start of the silent period is send over network as active packets. When sequential third silent packet is read from DPRAM, start of the silence period is decided. Packet type indicator which is written to the start of the internal network buffer for third silent packet is changed from character 'v' to the character 's' indicating the beginning of silence period and send over network. No other packet is send to host until the end of the silence period is decided. End of the silence period is decided when first active voice packet is read from the DPRAM in the silence period. By using packet type indicators 's' and 'v', receiver determines start and end of the talk spurts.

Voice activity detection can be enabled or disabled from the UART interface. One byte commands are assigned to enable and disable the VAD. When 'e' character is received from UART interface, VAD is enabled and when'd' character is received from UART interface VAD is disabled. When VAD is disabled every sample read from DPRAM is treated as active. By default VAD is disabled.

4.2.2.2 Receiver Functions

Receiver part of the software works against network delay variance, out of order packet reception and lost packets. Receiver part of the software sequences the received voice UDP packets, determines lost packets, and assigns playout time according to the adaptively determined playout buffer length at the beginning of each talkspurt.

As indicated in 4.1.3, microprocessor writes received voice samples from network to the DPRAM DAC data section. FPGA reads voice samples from DPRAM DAC data section and transfers them to the CODEC DAC section for playout. Every received packet is buffered before playing out to compensate the network delay jitter. Playout buffer length is determined using the adaptive playout buffering Algorithm 3, which is explained in Chapter 2 This algorithm is chosen according to the simulation results given in Chapter 5.

Playout buffer length is determined adaptively because of network delay characteristics changes from time to time. For every received packet, network delay and its variance is estimated adaptively from packet's send time and receive time. Playout buffer delay is determined from these calculated values as in equation 2.1, at the start of the each talk spurt. Playout buffer delay is changed from talk spurt to talk spurt.

For voice packets, packet type identification characters 'v' and's' are used as given in Table 4.4. Packet type identification character's' indicates the end of a talk spurt. Playout out buffer length is changed when received character is's'. For the first talk spurt playout delay of 18 ms is used. Playout buffering algorithm 3 given in section 2.3.1.3 is used for playout buffer delay calculation. This algorithm is chosen according to the MATLAB simulation results given in Chapter 5.

Network delay is calculated for each received packet using the following equation.

Network Delay = Receive Time – Send Time –Clock Offset

Clock offset is used to synchronize the two communication clients as explained in the clock offset calculation section 4.2.1.2.

In playout buffering algorithm 3, network delay estimate is the minimum of the network delay's observed by the packets within the talk spurt as given in equation 2.3. Therefore, network delay of the first packet of the talk spurt is recorded as the *minimum network delay* and this value is compared with the *network delay* of the subsequent received packets within the talk spurt. If a lower delay value is observed minimum network delay value is changed to this value. Also, for each received packet *network delay variance* is updated using the equation 2.2.
At the end of the talk spurt, which is decided with the reception of a packet with packet type identification character 's', *network delay* and *network delay variance* are used to calculate the playout time of the following talk spurt according to the equation 2.1 which is rewritten below using the software variable names and clock offset correction is included.

Playout time = (Send Time–Clock Offset) + Minimum Network Delay+4xNetwork Delay Variance

When the connection is established, microprocessor software informs FPGA that connection is established. A line connected from a pin of microprocessor to the FPGA accomplishes this. Microprocessor records the value of the timer as C*onnection Start Time.*

When connection is established FPGA sets its READ address pointer to the start of the DPRAM DAC data section and reads one DPRAM word every 0.5 ms. As mentioned in section 4.1.3. DPRAM DAC Data section is used as a circular buffer for playout buffering. A graphical illustration of DPRAM DAC data section implemented as a circular buffer and relative locations of the FPGA read pointer and microprocessor write pointer is seen in Figure 4.14.

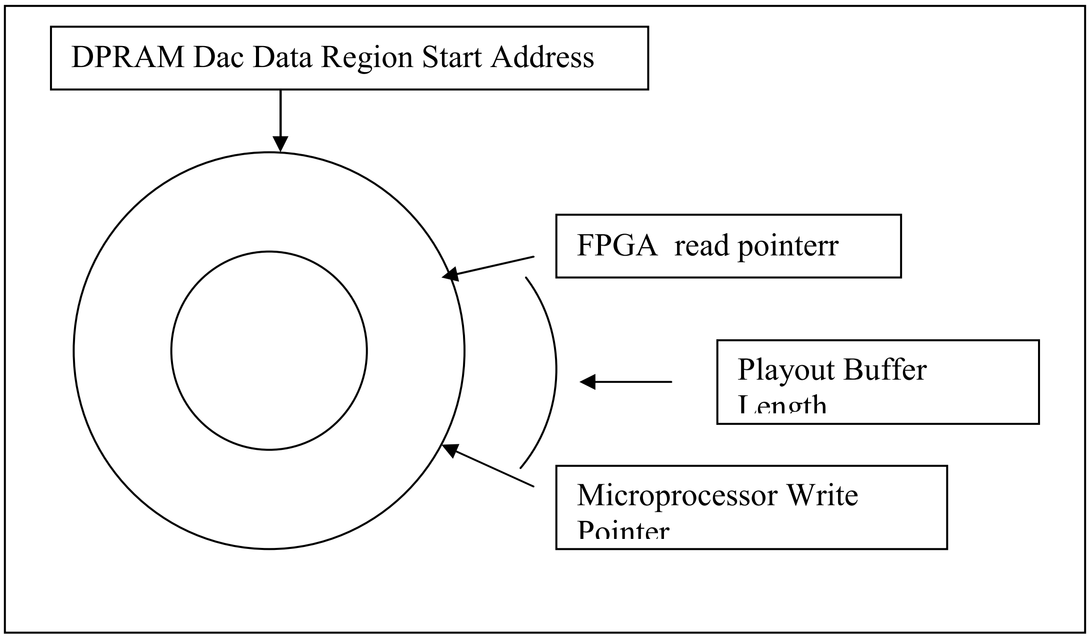
DPRAM Dac Data Region Start Address
FPGA read pointerr
Playout Buffer Length
Microprocessor Write Pointer

Figure 4.14: DPRAM DAC Section Implemented as a Circular Buffer.

DPRAM address, where calculated *playout time* corresponds, is further than DPRAM DAC data section start address by an amount of twice the difference between *Playout Time* and *Connection Start Time.* Since at the connection start time FPGA read address pointer is at the DPRAM DAC data section start address and increments by one every 0.5 ms and playout time is calculated using the unit of 1 millisecond. Playout Buffer length is the twice the difference between *playout time* and current value of the timer.

Voice samples extracted from the first voice UDP packet of the talk spurt are written sequentially starting from the DPRAM address where calculated playout time corresponds. Voice samples extracted from the following packets of the talk spurt are written sequentially after the first packet according to the sequence numbers. Lost packets are determined from the sequence numbers and DPRAM section where lost packets should be written is filled with zeros.

For each received UDP packet, with packet type identification character ‘v’, microprocessor records the current timer value as packets *receive time.* Then, it reads the one byte packet type identification character, 2-byte sequence number, and 4 byte *send time* timestamp from the received packet.

Then, network delay is calculated according to the equation 2.3. Calculated *Network Delay* and *sequence number* of each received packet is send from UART interface for diagnostics purposes. Network delay is compared with the previous minimum of the network delay. If it is smaller, minimum network delay value is changed. Then n*etwork delay variance* is updated using equation 2.2.

Number of lost packets is determined from the sequence number of the previously received packet and received packet. If the difference between the sequence numbers of the previously received packet and received packet is greater than one there is lost packets. Microprocessor fills DPRAM locations with zero corresponding to the lost packets, starting from the current microprocessor write address pointer location. Then, microprocessor writes the voice samples extracted from the received packet to the DPRAM DAC section by incrementing the pointer location by one for every four samples until the all 128 samples that a voice packet contains are written.

At the end of a talkspurt, which is decided with the reception of UDP packet with packet type identification character ‘s’, playout time is calculated according to the equation 2.1 and playout buffer length is determined. Then, microprocessors write pointer location is incremented or decremented according to the difference with the previously used buffer length to set the playout buffer length to the length that corresponds to the calculated new playout buffer delay.

4.3 User Interface Program

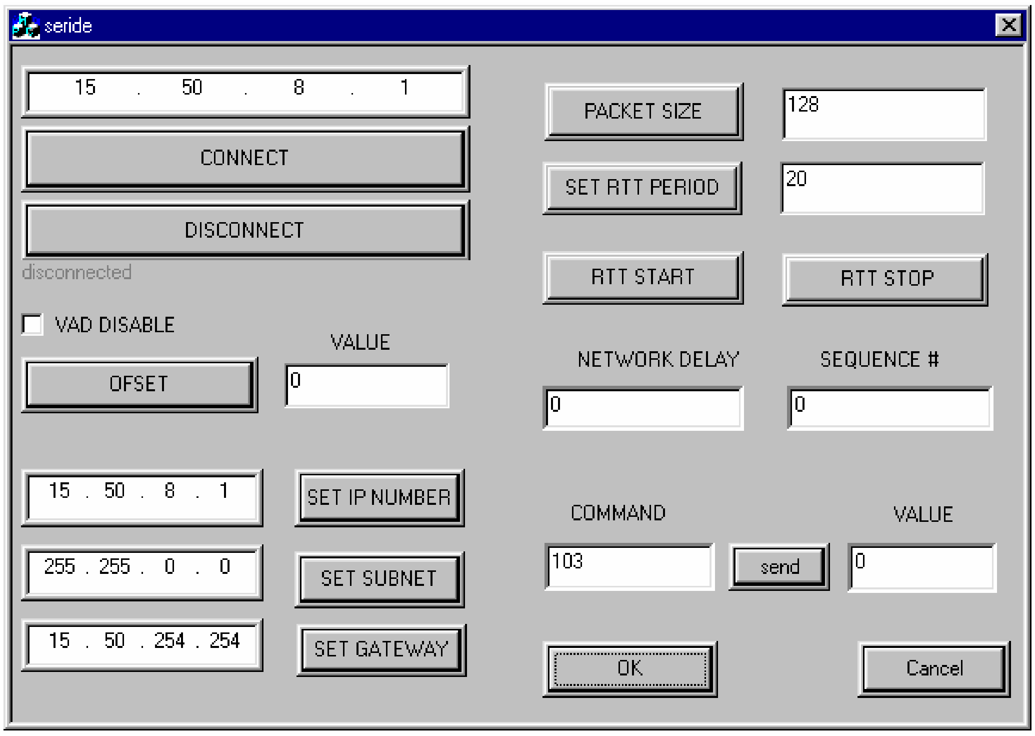

Figure 4.15: User Interface Program

In figure 4.15, user interface program is shown. By using this program user can send defined set of commands given in Table 4.4 to the developed VOIP device and can log the results of the measurements made by the VOIP device. User interface program is developed using the Microsoft Visual Studio 6.0 and runs on a windows based PC.

PC and VOIP device connection is established by connecting the COM1 port of the PC to the VOIP device serial port. Configuration of two communicating VoIP device is seen in Figure 4.16

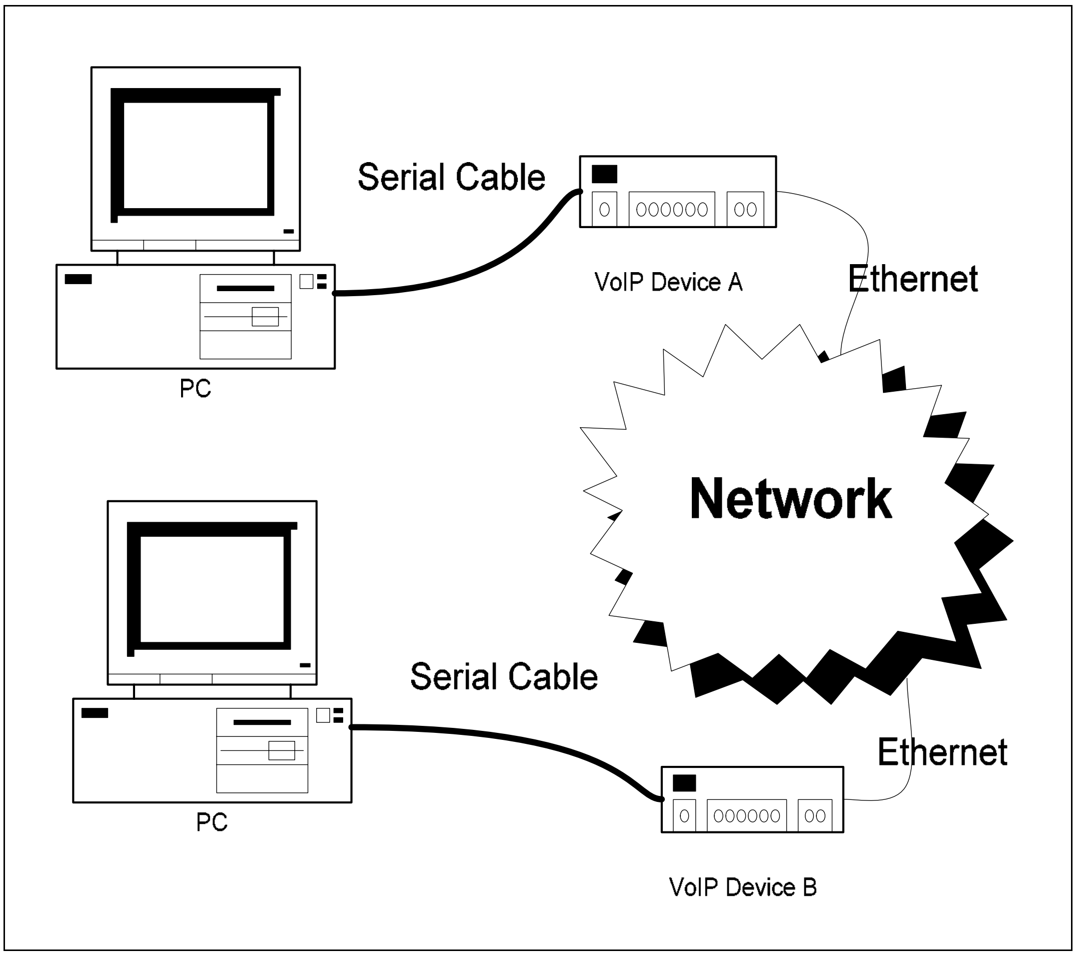

Figure 4.16: Two communicating VoIP Device.

As mentioned in the section 4.2.1.1, when device is inserted in a network, its IP number, subnet mask and gateway number settings should be done. These numbers should belong to the inserted subnetwork. These settings can be done using the buttons named *SET IP_NUMBER, SET SUBNET, SET GATEWAY* and their related edit boxes located at the left of the corresponding button as shown in Figure 4.15. When setting these numbers, first number should be written to the edit box and after corresponding button should be clicked. After network number settings are done, device is ready to use for VOIP communication.

Connection can be established with the host using the user interface program. Before connecting to a host clock offset with the host clock should be removed. For this purpose host IP number should be written to the edit box at the top left corner of the window. After host IP number is written, *CLOCK OFFSET* button should be clicked. Measured clock offset value is returned to the box which is on the right of the clock offset button. This value is recorded to the log file.

After clock offset is removed, connection can be established with the host by clicking the connect button. When connection is established, diagnostic message showing the status of the connection changes from *disconnected* to the *connected.* This message is located below the disconnect button.

When connection is established, VOIP device returns the sequence number and measured network delay of each received packet to the user interface program. User Interface program prints this values to the boxes named *sequence #* and *Network Delay*. These values are also written to the log file. User can end the connection by clicking the disconnect button. When connection is released, diagnostic message showing the status of the connection changes from connected to the disconnected.

As mentioned in section 4.2.2.1, VAD is disabled by default. By clicking the VAD Enable check box, VAD can be enabled. When VAD is enabled, VOIP device senses whether or not the user is speaking and sends voice packets only when user is speaking. User can disable the VAD by clicking the check box again.

Also, user can start round trip delay calculation and set round trip delay calculation parameters.

User can set size of the packets used in the round trip delay calculation by writing the packet size to the edit box, which is at right of the *PACKET SIZE* button, and clicking the *PACKET SIZE* button. Unit of the packet size is byte. Default value of the packet size is 128 bytes.

Also, time difference between departure times of the two successive packets used for round trip delay can be set by writing the value to the edit box, which is at the right of the *SET PERIOD,* and clicking the *SET PERIOD* button. Unit of the Round Trip Time (RTT) packet generation period is in milliseconds. Default value of the RTT packet generation period is 20 milliseconds.

After RTT measurement parameters are set, user can start RTT measurement by clicking the *RTT Start* button and end measurement by clicking the *RTT Stop* button. Measurement results returned from VOIP device are printed to the *Sequence* # and *Network Delay* boxes and also written to the log file.

4.4 Total End to End Delay

There are many contributors to the total end to end delay in VoIP systems. VoIP device is designed so as to minimize the total end to end delay. In the developed VoIP device, voice samples see many processes until they are transferred across Ethernet interface.

Main contributors to the total end-to-end delay in the developed VoIP device will be explained in this section. Intermediate processes inside the microprocessor will be ignored. A 100-clock cycle process inside the microprocessor costs 0.83 microseconds because of the high clock rate of 120 MHz.

Main contributor to the end-to-end delay in VoIP device is introduced to collect voice samples from codec. 16 ms delay is introduced for voice sample collection, since a voice packet consists, voice samples of 16 ms of duration. In the voice sample collection process, samples extracted from the coded are written to the DPRAM ADC data section as explained in section 4.1.5. Since this process is a pipelined process, there is no extra delay introduced on voice samples to write them to the DPRAM.

After voice samples are written to the DPRAM, microprocessor is interrupted. In the interrupt subroutine of the microprocessor, microprocessor reads voice samples from DPRAM. Voice samples of 16 ms in duration are located in 32 subsequent locations in DPRAM where each location consist 4 samples. Microprocessor reads single location, compares samples with VAD threshold and then writes samples to the network buffer. Then it advances to the next location in the DPRAM. It is measured experimentally that, microprocessor reads single location of DPRAM every 1 microseconds in the interrupt subroutine. Since there are 32 locations, delay introduced in the interrupt subroutine is 32 microseconds. This relatively low value when compared to the voice sample collection delay of 16 ms.

After all samples are written to the network buffer, packet is send via Ethernet interface and network delay is introduced on the packet.

At the receive side, when a voice packet is received, its playout time and corresponding DPRAM location is calculated and received voice samples are written to that DPRAM location. This process is very similar to the transmitter side and delay introduced by this process is very low compared to the network delay and voice sample collection delay. There is also playout buffer delay, which is used to compensate the variable part of the network delay and calculated by playout buffer algorithm.

We can say that, main contributors to the total end to end delay are voice sample collection delay, network delay and playout buffer delay. Processing delay introduced by developed VoIP device is on the order of microseconds and not comparable to the main contributors of the delay.

4.5 VoIP Device Compatible Computer Software Development

In this section, basic C language programming techniques will be described in order to write software, which runs on PC platforms, that is compatible with the packet structure of the developed VoIP device. Implementation issues that are same with the VoIP device microprocessor software are not given. Functions that are specific to the main software structure and usage of the network adapter card and soundcard will be given.

4.5.1 The basic system design

The program has to have three main modules. These modules are named transmitter, receiver and control modules. This system structure is like microprocessor software structure of the VIP device. Basic system structure is shown in Figure 4.17.

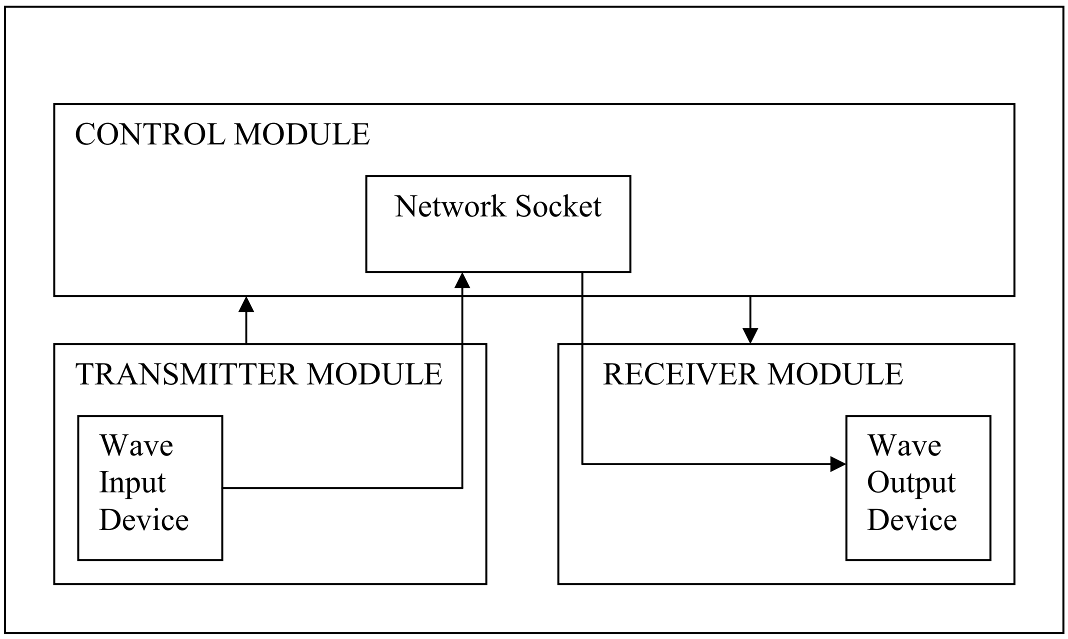

Figure 4.17: Basic system structure of the VoIP software on a PC platform

To run three modules of the software in parallel, “thread” structure of the “C” programming language can be used. Threads work as a separate programs in parallel. Codes for the three modules should be written as different programs. Commands for thread are included in the *process.h* header file. Threads can be started to work using *_beginthread(*(threadName), 0, Null)* command. After this command is executed thread function runs as an separate programs. Communication between threads is performed using *event* structures. Threads can set or reset events using S*etEvent(handle)* and *ResetEvent(handle)* commands respectively. A thread can wait for a event using *WaitForSingleObject(handle, duration)* command. This command pauses the thread execution until *handle* event is set by another thread.

As explained, transmitter, receiver and control modules can be run in parallel using thread structure and they can communicate with each other using event structure. In the following sections all three modules will be explained.

4.5.2 Control Module

Control module is the main part of the software. It controls the execution of the other two modules (threads), creates a UDP socket, and controls connection establishment process.

First of all, control module should create a socket to listen a UDP port for connection requests. A UDP socket can be created executing following command;

listen_socket = socket(AF_INET, SOCK_DGRAM,0).

After then, local address and port combination should be combined with the created socket. *Bind()* command can be used for this purpose. Before using this command a *sockaddr_in* structure should be created to identify the local address and port. This structure can be created using the code segment seen in Figure 4.18.

```
Struct sockaddr_in local;

local_sin_family =AF_INET;

local.sin_addr.s_addr= ip_address;

local.sin_port = port_number;
```

Figure 4.18: Code segment to create sockaddr_in structure

After *sockaddr_in* structure is created; it can be used in *bind* command to associate local address and port combination with the *listen_socket* in the following way;

bind(listen_socket, (struct sockaddr)&local, sizeof(local));*

After then, listen_socket can be used to listen the incoming packets. Another sockaddr_in structure should be defined for the host port and address. Control module listens network by executing following command.

retval = recvfrom (listen_socket, PacketBuffer ,sizeof (Buffer),0,
(struct sockaddr)&host, &hostlen);

PacketBuffer contains the UDP payload of the received packet and *retval* shows the received number of bytes. To send a packet to the host a similar function *sendto* should be used .

sendto (listen_socket, SendBuffer ,sizeof (SendBuffer),0,(struct sockaddr)&host, &hostlen);

Control module checks every received packet's packet type identification character as in the developed VoIP device. List of packet type identification characters are shown in Table 4.4. Developed software should take similar actions with the developed VoIP device for each received packet type.

After connection is established by performing the same packet exchange structure in the VoIP device, streaming voice packets can be received or transmitted. Control module starts the receiver and transmitter modules threads using *_beginthread()* command. When control module receives connection close packet it ends the operation of the receiver and transmitter threads using *_endthread()* command.

Voice packet reception is done in the control module; since control module has to check packet type identification character of the each received packet to decide which action to take. If a voice packet is received by the control module, it processes the RTP header as VoIP device, assigns playout buffer length for received voice samples. Then writes the received packets to the previously allocated playout buffer according to the assigned playout buffer length. Receiver module (thread) then plays samples contained in the playout buffer in a periodic manner.

4.5.3 Receiver Module

Receiver module plays the audio written to the playout buffer by the control module. Computer soundcard wave out capabilities are used for this purpose. *WAVEFORMATHEX* structure contained in the windows library is used to set the properties of the played audio. First a variable with the type *WAVEFORMATHEX* should be defined and audio properties should be set. Code segment to set the audio properties to the 8 kHz PCM format is shown in figure 4.19

```
WAVEFORMATHEX waveFormat;
waveFormat.wFormatTag = WAVE_FORMAT_PCM;
waveFormat.nChannels = 1;
waveFormat.nSamplesPerSecond = 8000;
waveFormat.wBitsPerSample=8;
waveFormat.nBlockAlign=waveFormat.nchannels*
(waveFormat.wBitsPerSample/8);
waveFormat.nAvgBytesPerSecond=waveFormat.nChannels*
waveFormat.nBlockAlign;
waveFormat.cbSize = 0;
```

Figure 4.19 : Code segment t set the soundcard waveout properties.

Then a WAVEHDR structure should be created to assign a buffer to the wave out device. Buffer named *waveoutbuffer* is associated with the *waveheader* by executing the following command.

Waveheader.lpData = waveoutbuffer;

After soundcard settings are performed, receiver thread continuously reads the playout buffer and writes the samples to the *waveoutbuffer* to be played out by the soundcard. To play the voice samples contained in the *waveoutbuffer* , *waveheader* should be prepared and associated with wave out device. Following command should be executed for this purpose.

waveOutPrepareHeader(outHandle,&waveheader,sizeof(WAVEHDR));

OutHandle is a structure with type *HWAVEOUT* and this structure handles the waveout device. After *waveheader* is prepared *waveoutbuffer* can be played by executing the following command.

waveOutWrite(outHandle, &waveheader, sizeof(WAVEHDR));

Status of the waveout device can be viewed by checking *waveHeader.dwFlags.* When wave out device plays all the samples contained in the *waveoutbuffer* value of the *waveHeader.dwFlags* is set to 3. After all the samples contained in the *waveoutbuffer* are played, waveheader should unprepared by executing *waveOutPrepareHeader* command. Then new samples should be read from the playout buffer and written to the *waveoutbuffer* to be played out by the soundcard. Same commands explained up to now should be executed again to play out the samples contained in the *waveoutbuffer.* These commands should run in loop structure that periodically fills the *waveoutbuffer* with the received voice samples from the playout buffer and plays out the voice samples using the soundcard for the periodic reconstruction of the voice.

4.5.4 Transmitter Module

After connection is established, transmitter module uses the computer soundcard wave in capabilities to extract the user voice samples. Extracted samples are packetized according to the packet structure of VoIP device and then transmitted to the host using network adapter card.

Soundcard should be configured as wave input device and audio properties should be set as in Figure 4.19. *WAVEHDR* structure should be created to assign a buffer to the wave in device. Then an input buffer, with the size of 128 bytes should be created and associated with the *waveheader* as in the receiver module. Buffer size is 128 bytes; because VoIP device voice packets contain 128 bytes of voice samples that correspond to the 16 ms of voice in duration. A wave input device handle should be created with the structure type *HWAVEIN.* Created *waveheader* should be associated with the wave input device using the following command.

waveInPrepareHeader(InHandle,&waveheader,sizeof(WAVEHDR));

Next, buffer associated with the waveheader should be added to the wave input device by executing following command.

waveInAddBuffer(InHandle,&waveheader,sizeof(WAVEHDR));

Then, wave input device can be started to sample incoming voice by executing *waveInStart(inHandle)* command. Status of the wave input device can be viewed by checking *waveHeader.dwFlags.* When wave input device fills the 128 bytes long input buffer with voice samples value of the *waveHeader.dwFlags* is set to 3. This flag can be used to decide when input buffer is filled. When input buffer is ready with vice samples waveheader should be unprepared using the following command;

waveInUnPrepareHeader(InHandle,&waveheader,sizeof(WAVEHDR));

Voice packet should be created in voice packet buffer using voice samples collected in the input buffer. Voice packet buffer should contain RTP header, created as in the

VoIP device and the extracted samples from the soundcard wave input device. When voice packet is created, it should be sent to the host over network using computer network adapter card. UDP socket created in the control module can also be used for packet transmission. Voice packet should be sent to the host using the following command;

sendto (listen_socket, SendBuffer ,sizeof (SendBuffer),0,(struct sockaddr)&host, &hostlen);

Transmitter thread should run in an continuous way until thread is end by control module when connection is closed. Therefore, after packet is send to the host, transmitter module should return to the beginning to create another voice packet and send to the host in a loop structure.

CHAPTER 5

ADAPTIVE PLAYOUT ALGORITHM SIMULATIONS AND COMPARISONS

The performances of the four adaptive playout buffering algorithms, explained Chapter 2, are needed to be evaluated to decide which algorithm to use in the developed VoIP device.

These algorithms can be compared with each other by actually implementing all of the algorithms on the different versions of the device or by evaluating their performance with the simulations. But actual implementations of the algorithms do not give the chance of evaluating the algorithms in the identical network conditions. Since the network characteristics are very dynamic, same experimental conditions cannot be created for all of the algorithms. Therefore it is preferred to use simulations to compare algorithms.

Simulations can be carried on the traces that are artificially generated on the simulation environment, which is the MATLAB in our case or with the real traces collected by observing the network. We have carried simulations on both artificially generated traces and network-collected traces. By this approach we are able to run all four algorithms on same set of traces and we were thus able to compare the performance of the algorithms under identical network conditions. Developed VoIP device is used the collect the network traces that will be used in the simulations. For this purpose, test setups are developed in the METU and Hacettepe University.

Traces are collected for two days using the network delay and sequence number logging feature of the User Interface Program of the VoIP device. VoIP device sends the sequence number and corresponding measured delay values for the received packets on a VoIP session to the user Interface Program. And user interface program writes these values to a log file.

In the next section, collected network traces and calculated playout times for the algorithms, Algorithm 1 through Algorithm 4, referenced in Chapter 2 are given.

5.1 Simulations

In this section, simulation results of the algorithms for the three traces collected using VoIP device are given. Experiments are carried between METU and Hacettepe University.

Trace network delay values, collected by VoIP device, are added to the send time, which is generated according to the sequence numbers corresponding to network delays in the log file, to calculate the receive time of the packets. Algorithms use Send time and Receive Time, as timestamps in an actual implementation and calculate playout time. Send time of the first packet is taken as 0 end following packet send times are calculated according to this time origin.

Trace 1 is collected at 13:15 PM at 29.08.2003. Network delay values for collected trace are given in Figure 5.1. Mean network delay value for the trace is 37 ms, minimum value is 16 ms and the maximum value is 71 ms. 8.37% of the packets are lost in the network. Probability density function of Trace 1 is extracted by normalizing the histogram of the Trace 1 with packet count and it is shown on the Figure 5.2. It is noted that PDF of the Trace 1 follows a gamma distribution with parameters 40.19 and 0.92. Gamma distribution with these parameters is also plotted on Figure 5.2. Detailed information on gamma distribution can be found in appendix B.

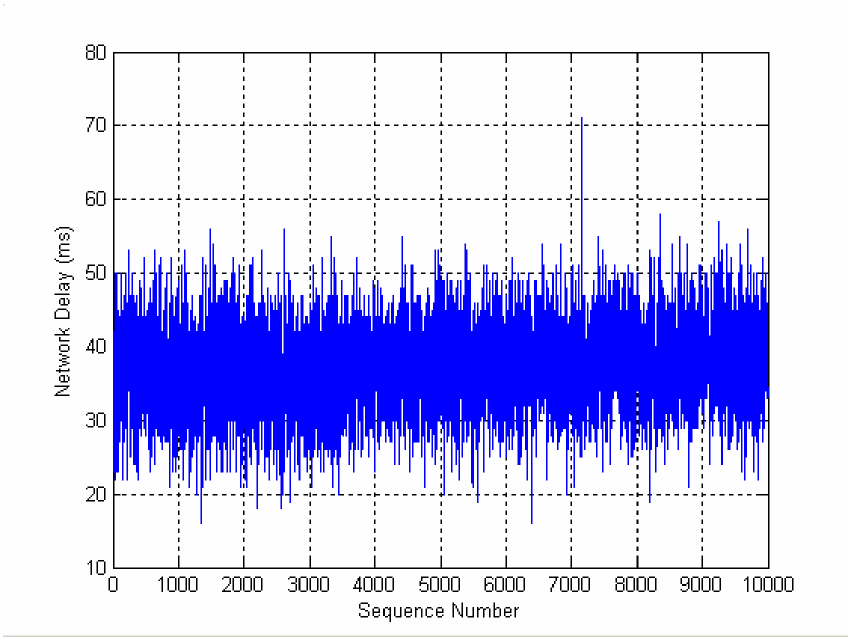

Figure 5.1: Delay Measurement Result for Trace 1

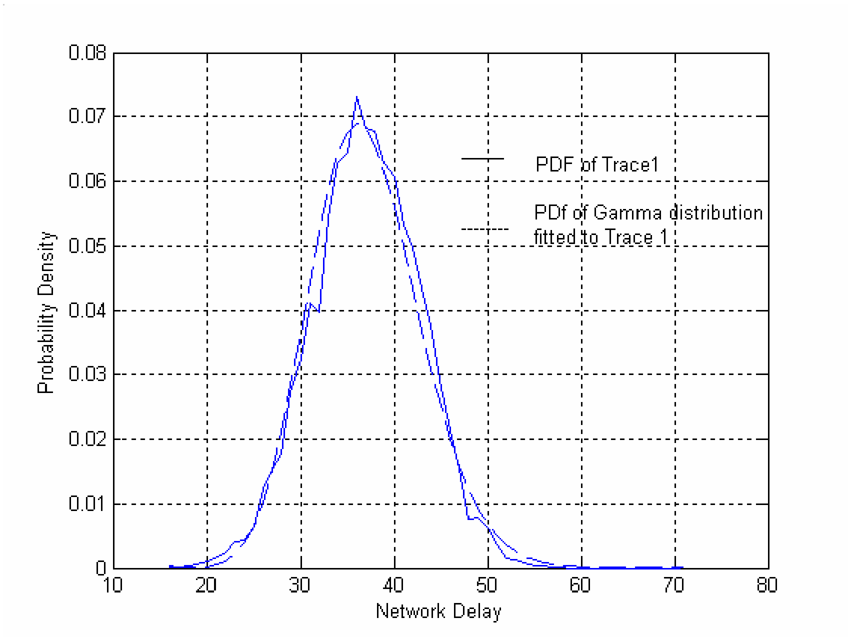

Figure 5.2: PDF of Trace1 and Gamma Distribution Fitted to Trace 1

Figures 5.3 through 5.6 shows send time, receive time and corresponding playout time settings of Algorithms 1 through 4 for the Trace 1. Receive time later than corresponding playout time shows late arrival for a packet and that packet is treated as lost.

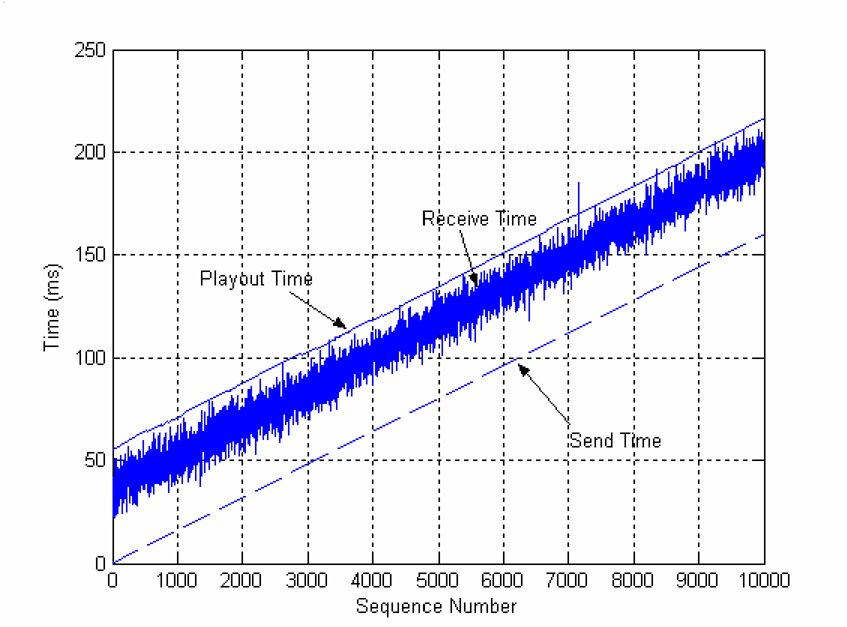

Figure 5.3 : Calculated Playout Time by Algorithm 1 for Trace 1

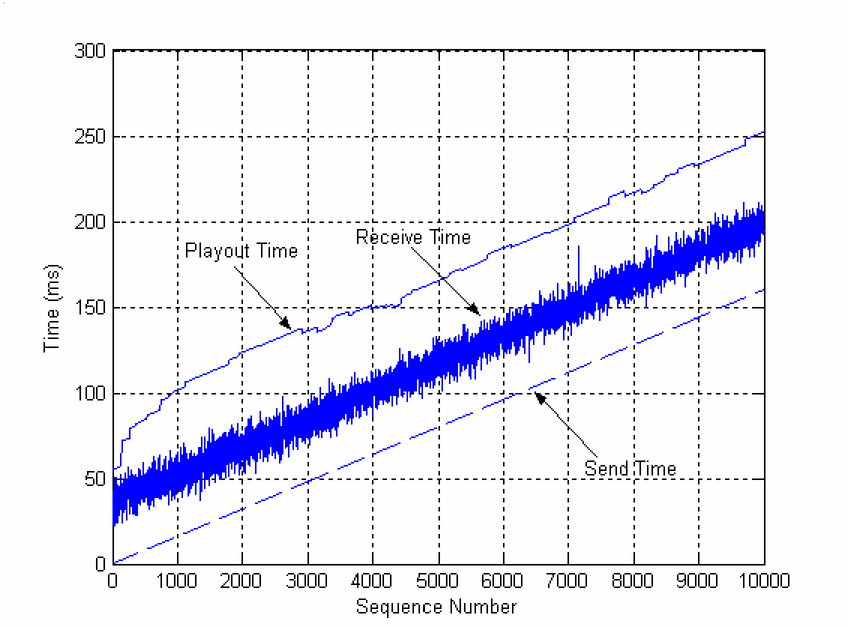

Figure 5.4 : Calculated Playout Time by Algorithm 2 for Trace 1

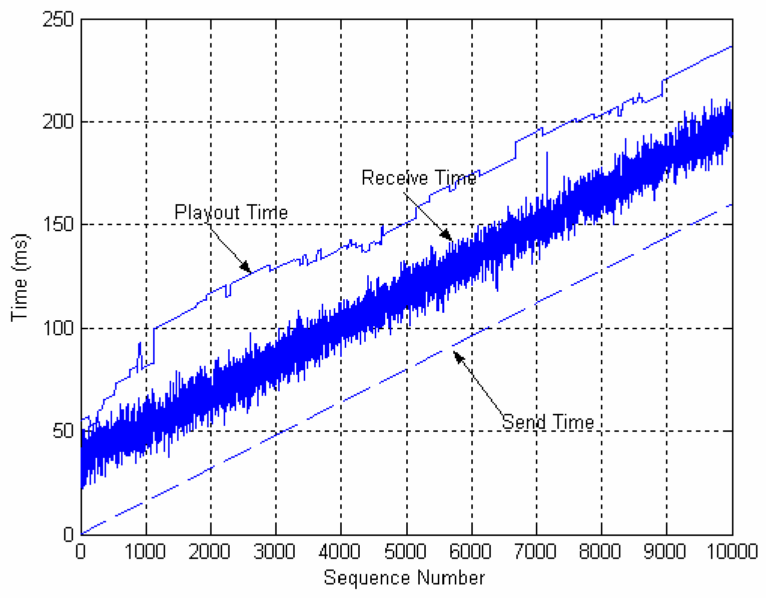

Figure 5.5 : Calculated Playout Time by Algorithm 3 for Trace 1

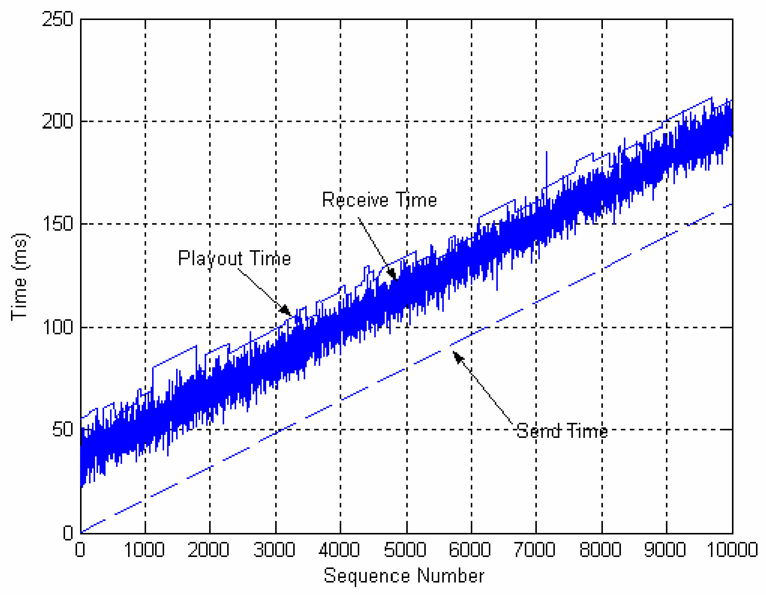

Figure 5.6 : Calculated Playout Time by Algorithm 4 for Trace 1

Number of lost packets due to the late arrival after their scheduled playout time and mean playout delay calculated for the algorithms 1 through 3 is given in Table 5.1 for Trace 1.

Table 5.1: Mean Playout Delay and Number of Lost Packets for Trace 1

	Algorithm 1	Algorithm2	Algorithm3	Algorithm 4
Number of Lost Packets Due to Late Arrival	8	0	2	101
Mean Playout Delay (ms)	55.189	86.73	75.77	52.92

Network delay values for Trace 2 are given in Figure 5.7. Trace 2 is collected at 10:45 AM at 29.08.2003. Mean network delay value for the trace is 33.52 ms, minimum value is 7 ms and the maximum value is 57 ms. 8.04% of the packets are lost in the network. Probability density function of Trace 1 is extracted by normalizing the histogram of the Trace 1 with packet count and it is shown on the Figure 5.8. It is noted that PDF of the Trace 1 follows a gamma distribution with parameters 24.81 and 1.35 (std=6.49). Gamma distribution with these parameters is also plotted on Figure 5.8.

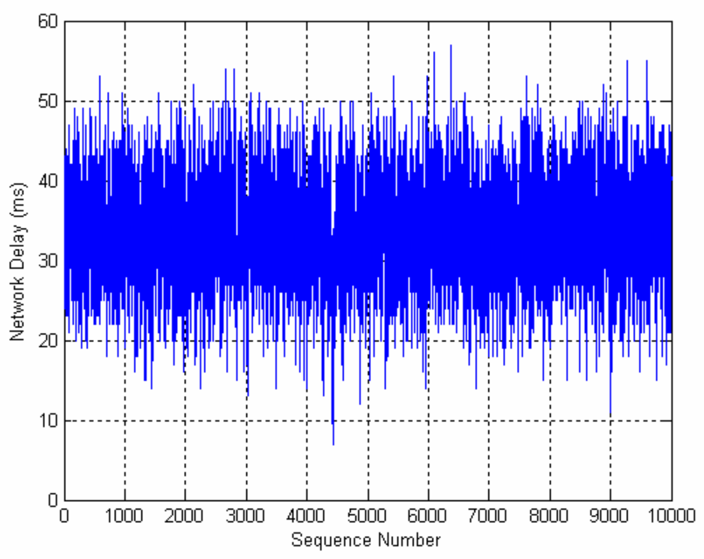

Figure 5.7: Delay Measurement Result for Trace 2

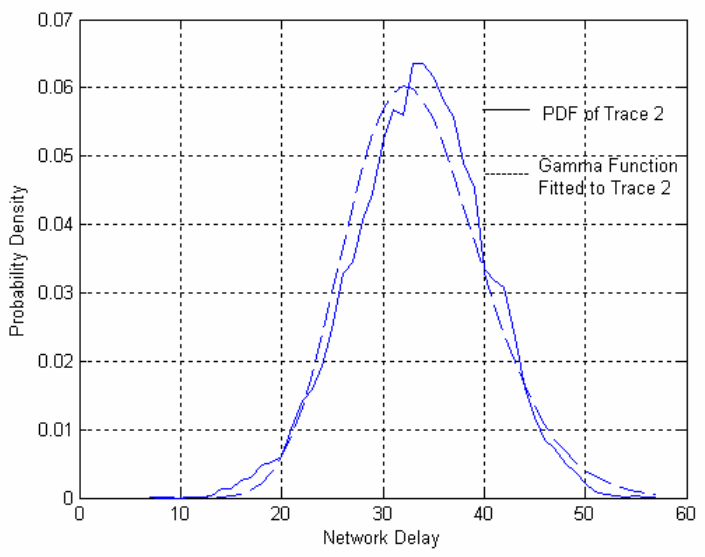

Figure 5.8: PDF of Trace1 and Gamma Distribution Fitted to Trace 2

Figures 5.9 through 5.12 shows send time, receive time and corresponding playout time settings of Algorithms 1 through 3 for the Trace 1.

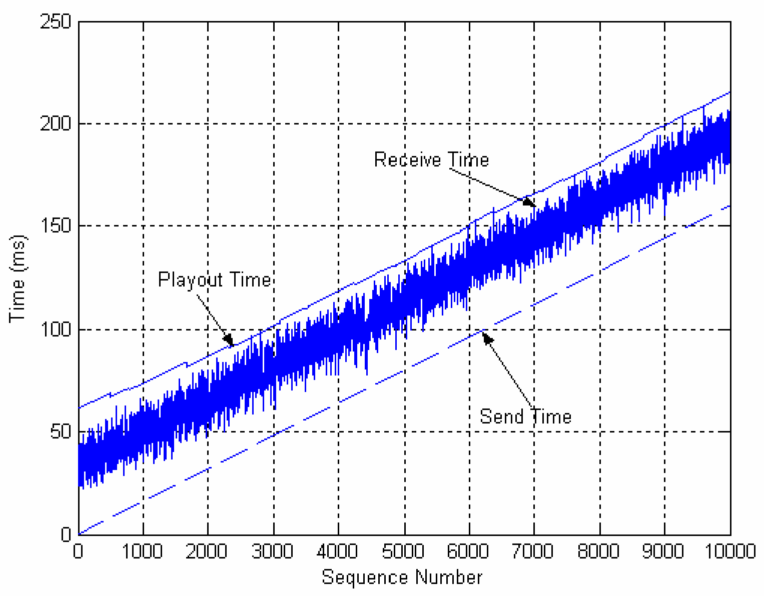

Figure 5.9 : Calculated Playout Time by Algorithm 1 for Trace 2

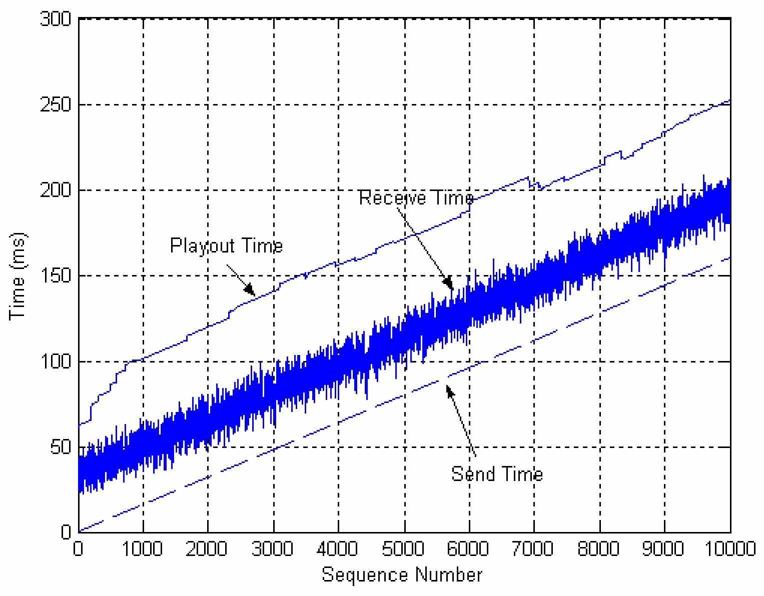

Figure 5.10 : Calculated Playout Time by Algorithm 2 for Trace 2

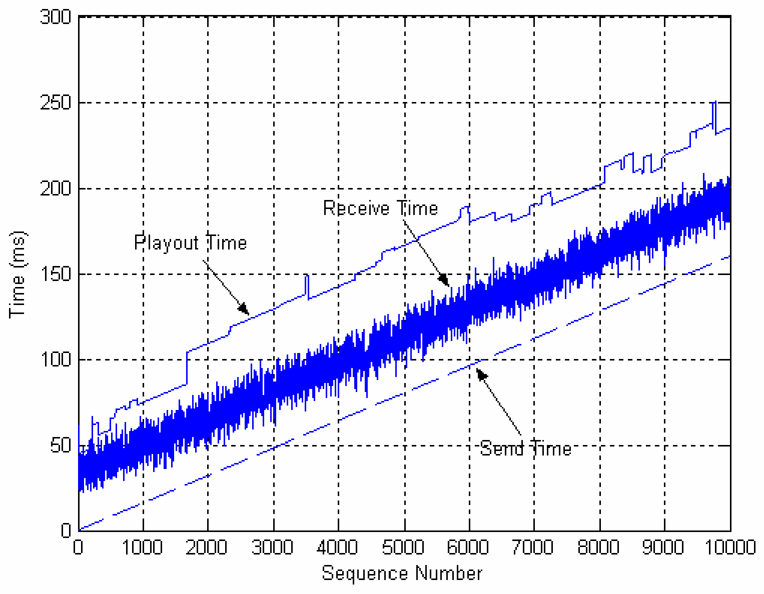

Figure 5.11 : Calculated Playout Time by Algorithm 3 for Trace 1

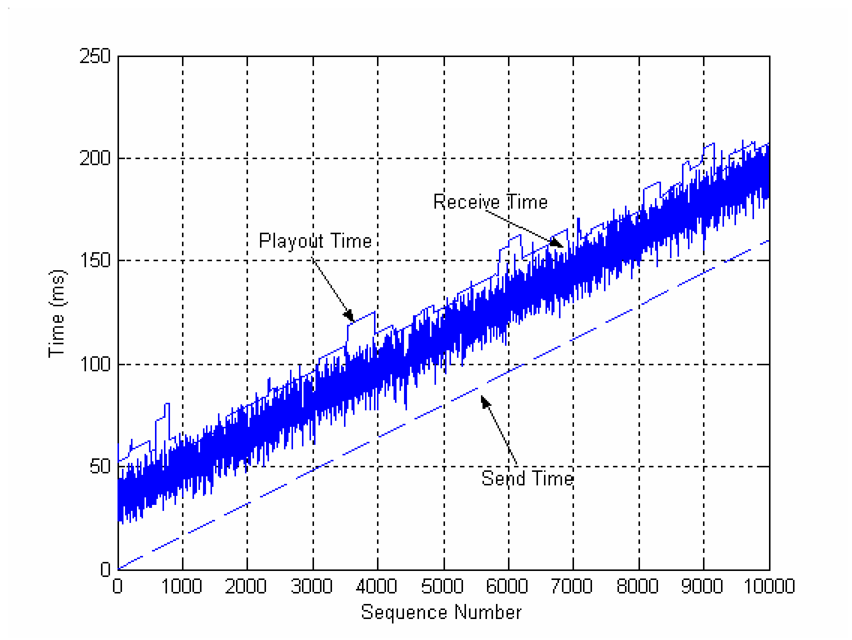

Figure 5.12 : Calculated Playout Time by Algorithm 3 for Trace 1

Number of lost packets due to the late arrival after their scheduled playout time and mean playout delay calculated for the algorithms 1 through 4 is given in Table 5.1 for Trace 1.

Table 5.2: Mean Playout Delay and Number of Lost Packets for Trace 2

	Algorithm 1	**Algorithm2**	**Algorithm3**	**Algorithm 4**
Number of Lost Packets Due to Late Arrival	7	0	5	154
Mean Playout Delay (ms)	54.84	89.06	76	50.62

5.2 Comparison and Discussion of results

As we can see from the simulation results for Trace 1 and Trace 2, Algorithm 4 perform the playout time setting with smaller mean playout delay than other algorithms. But number of lost packets, due to late arrival, is bigger for Algorithm 4.

This is due to the fact that, Algorithm 4 is designed for networks that network delay spikes are seen often. But for our traces, network delay spikes are not seen too much. Also, as given in [2], constant parameters used in Algorithm 4 are very sensitive to the network characteristics.

Algorithm 2 gives the biggest mean playout delay between all three algorithms. But it gives the minimum number of lost packets.

As seen from the figures for playout times, Algorithm 1 sets the playout time smoother than other algorithms. This results in more preservation of actual silence period lengths. Preservation of silence periods also increases the perceived quality of the played out speech.

As mentioned, Algorithm2 gives the minimum number of lost packets and algorithm 4 gives the minimum mean playout delay for the given traces. But Algorithm 1, gives slightly higher number of lost packets than other algorithms and gives smaller mean playout delay value. It is seen that, for the given traces, by considering both number of lost packets and mean playout delay, Algorithm 1 performs better than other algorithms. Also silence period compression for algorithm 1 is minimum, which also increases perceived quality of played out speech. Algorithm3 has advantage of ease of implementation where network delay estimate is calculated by finding the minimum of the observed network delays in a talkspurt. This algorithm gives smaller number of lost packets at the expense of higher mean playout delay.

If we look at the statistics of the collected traces, they have both number of lost packets almost %8. This is relatively high percentage that degrades the quality of the played out speech. Therefore, we should choose the algorithm that gives small number of lost packets. Algorithm 1semms most appropriate algorithm, but it has a high computational complexity because of. On the other hand Algorithm 3 achieves this goal, and it has a ease of implementation. Algorithm 3 is chosen to use in the developed VoIP device and implemented on microprocessor software as explained in section 4.2.2.2.

CHAPTER 6

CONCLUSION AND FUTURE WORK

In the framework of this thesis, an embedded Internet telephony device, providing full duplex voice communication over internet has been developed. Various subjects come on to focus in the development cycle.

First, there was a need for a protocol to use as network transport layer protocol in the developed device. Existing protocols, such as TCP and UDP are examined, their advantages and disadvantages, when used in a real time application, are discussed. Because of their lack of transmitting timing information and other disadvantages explained in Chapter 2, RTP, latest protocol used in real time applications, [4], is examined. And finally a RTP like custom protocol is developed and used in the developed device as transport layer protocol. This protocol satisfies all the requirements existing in an Internet Telephony application.

When we try to measure the one way network delay of the received packets, it is seen that there is a clock offset and clock skew problem. Clock offset problem is solved with the method given in [5]. This method removes the clock offset by assuming transmitted packets travels with the same network delay in both directions. This results in an error of a few milliseconds in the clock offset calculation and ear is tolerant to such a delay. Methods that give more accurate results can be used. For example both communication clients can be synchronized in the order of 100 ns by using Global Positioning System (GPS),[5]. However usage of GPS is not practical because of its high cost. Internet Telephony applications does not need synchronization in the order that GPS satisfies. GPS can be used in the systems where accurate one-way delay measurements are needed.

A VAD algorithm is developed and used in order to discriminate the talkspurt boundaries and to reduce transmission rate. Talkspurt boundaries should be determined since, adaptive playout algorithms sets playout time at the start of talkspurts.

Main problem in VoIP applications is network delay variance and packet loss. Lost packets are interpolated using zero insertion. Better QoS can be achieved by using more sophisticated interpolation methods or FEC methods can be used to increase the received QoS. Network delay variance is compensated using the one of the adaptive playout buffering algorithms presented in [3]. This algorithm is chosen according to the trace driven simulation results. Simulation approach is preferred in order to compare performance of the algorithms in identical network conditions. Simulation network delay traces are collected with the developed VoIP device in the experiments carried between Hacettepe University and METU.

A user interface program is developed to control the device and collect the measurement results such as network delay, and Round Trip Delay.

Developed VoIP device is fully tested in the experiments on local area networks of Aselsan Inc. and METU and also in the internet between Hacettepe University and METU. Device gives toll quality voice in local area networks. Perceived QoS decreases in wide area networks but still remains satisfactory because of high number of network related lost packets. Where packet lost rates between Hacettepe University and METU is on the order of almost 8% measured.

As future work, network delay and round trip delay measurement facility of the device can be used in the experiments to extract the characteristics of networks.

These experiments require extensive measurement of the network delay to be able to characterize the network with statistical approaches. Perceived QoS in VoIP communication can be tested between long distances where network delay and its variance are higher. Adaptive Playout Buffering algorithm simulations can also be done on these traces to compare the algorithms. Also, FEC algorithms can examined and one of them can be embedded on microprocessor to compensate lost packets. Advanced speech coding techniques can be used in order to reduce the transmission rate.

REFERENCES

1. Sue B. Moon, “Measurement And Analysis Of End-to-End Delay and Loss In The Internet”, University of Massachusetts Amherst, February 2000

2. J.Rosenberg, L. Qui, H. Schulzrinne, “Integrating Packet Fec into Adaptive Voice Plyout Buffer Algorithms on the Internet”, Infocom 2000

3. R. Ramjee, J. Kurose, D. Towsley, H. Schulzrinne, “Adaptive Playout Mechanisms for packetized Audio Applications in Wide Area Networks”, Proceedings of the Conference on Computer Communications (IEEE Infocom), Toronto, Canada, 1994

4. A.S. Tanenbaum, “Computer Networks 4th Ed”, Prentice-Hall, 2003

5. K. Fujimoto, Adaptive Playout Buffer Algorithm for Enchancing Perceived Quality of Streaming Apllications, Osaka University, Japan, February 2002

6. M. Narbutt, L. Murhpy, Adaptive Playout Buffering For Audio/Video Transmission Over the Internet, University College Dublin, Dublin, Ireland,2000

7. V. Jacobson, “Congestion Avaoidance and Control”, Proc. 1988 ACM SIGCOMM Conf., pages 314-329, August 1988

8. Jon Postel, editor, “Transmission Control Protocol Specification”, ARPANET Working Group Request For Comment, RFC 793, September 1981

9. D. Mills, “Internet Delay Experiments”, ARPANET Working Group Request for Comment, RFC 889, December 1983

10. Vern Paxson, “End-to-End Internet Packet Dynamics”, Network Research Group, Lawrence Berkeley Netional Laboratory, University of California, Berkeley, June 1997

11. J-C. Bolot, “End-to-End Packet Delay and Loss Behavior in the Internet,” *Proc. SIGCOMM '93*, pp. 289-298, Sept. 1993.

12. Jean-Chrysostome Bolot and Andres Vega Garcia, “Control mechanisms for packet audio in the Internet,” Proceedings of the Conference on Computer Communications (IEEE Infocom), San Fransisco, California,Mar. 1996

13. Sue Moon, Paul Skelly, and Don Towsley, “Estimation and removal of clock skew from network delay measurements,” in Proceedings of the Conference on Computer Communications (IEEE Infocom), New York, Mar. 1999

14. Sanghi, D., Gudmundsson, O., Agrawala, A., and Jain, B.N. “Experimental assessment of end-to-end behavior on Internet.”, Proceedings of INFOCOM ’93, pp. 867–874., 1993

15. Schulzrinne, H. “RTP profile for audio and video conferences with minimal control.”,RFC 1890, Internet Engineering Task Force, Jan 1996.

16. C.J. Sreenan, Jyh-Cheng Chen, Prathima Agrawal, and B. Narendran, “Delay reduction techniques for playout buffering,” *IEEE Transactions on Multimedia*, vol. 2, no. 2, June 2000.

Appendix A

Pseudo code for the *Send_Adc_Data_Task*

```
1.    Record timer value as packet generation timestamp.
2.    Increment sequence number by 1
3.    Allocate packet buffer
4.    Write packet type identifier character 'v' to the buffer
5.    Write sequence number to the buffer
6.    Write generation time timestamp to the buffer
7.    If  adc_data_section_identifer_pin = low
          ADC_data_Read_pointer =  Start of te ADC data section 1
      Else
          ADC_data_Read_pointer =   Start of te ADC data section 2
8.    For i=1:32 do
      {
          Read DPRAM location where  ADC_data_Read_pointer is indicating
          Increment  ADC_data_Read_pointer by 1
          Compare read 4 samples with VAD Threshold
          Increment ActiveSampleCount by 1 if samples are exceeding threshold.
          Write 4 samples read from DPRAM to the buffer
      }
9.    If ActiveSampleCount > 64
          Set packet VAD status as Active
      Else
          Set packet VAD status as Silent
10.   If packet Vad status = Silent
```

```
          Increment SuccessiveSilentPacketCount by 1
           Else
         SuccessiveSilentPacketCount = 0
11.       If VAD enabled
      {
         If   SuccessiveSilentPacketCount < 3
          Send the buffer to the host  using UDP protocol
      }
    Else
         Send the buffer to the host  using UDP protocol
```