5-4-2007

An Analysis of the MOS under Conditions of Delay, Jitter and Packet Loss and an Analysis of the Impact of Introducing Piggybacking and Reed Solomon FEC for VOIP

Alexander F. Ribadeneira

AN ANALYSIS OF THE MOS UNDER CONDITIONS OF DELAY, JIITTER AND PACKET LOSS AND AN ANALYSIS OF THE IMPACT OF INTRODUCING PIGGYBACKING AND REED SOLOMON FEC FOR VOIP

by

ALEXANDER F. RIBADENEIRA

Under the direction of Anu G. Bourgeois

ABSTRACT

Voice over IP (VoIP) is a real time application that allows transmitting voice through the Internet network. Recently there has been amazing progress in this field, mainly due to the development of voice codecs that react appropriately under conditions of packet loss, and the improvement of intelligent jitter buffers that perform better under conditions of variable inter packet delay. In addition, there are other factors that indirectly benefited VoIP. Today, computer networks are faster due to the advances in hardware and breakthrough algorithms. As a result, the quality of VoIP calls has improved considerably. However, the quality of VoIP calls under extreme conditions of packet loss still remains a major problem that needs to be addressed for the next generation of VoIP services. This thesis concentrates in making an analysis of the effects that network impairments, such as: delay, jitter, and packet loss have in the quality of VoIP calls and approaches to solve this problem. Finally, we analyze the impact of introducing forward error correction (FEC) Piggybacking and Reed Solomon codes for VoIP. To measure the mean opinion score of VoIP calls we develop an application based on the E-Model, and utilize perceptual evaluation of speech quality (PESQ).

INDEX WORDS: VoIP, FEC, SIP, MOS, E-Model, PESQ, RTP, RTCP.

AN ANALYSIS OF THE MOS UNDER CONDITIONS OF DELAY, JIITTER AND PACKET LOSS AND AN ANALYSIS OF THE IMPACT OF INTRODUCING PIGGYBACKING AND REED SOLOMON FEC FOR VOIP

by

ALEXANDER F. RIBADENEIRA

A thesis Submitted in the Partial Fulfillment of the Requirements for the Degree of

Master of Science

in the College of Arts and Sciences

Georgia State University

2007

AN ANALYSIS OF THE MOS UNDER CONDITIONS OF DELAY, JIITTER AND PACKET LOSS AND AN ANALYSIS OF THE IMPACT OF INTRODUCING PIGGYBACKING AND REED SOLOMON FEC FOR VOIP

by

ALEXANDER F. RIBADENEIRA

Major Professor: Anu G. Bourgeois

Committee: Raheem A. Beyah

Yingshu Li

Electronic Version Approved:

Office of Graduate Studies

College of Arts and Sciences

Georgia State University

May 2007

DEDICATION

To all the people who believe in social justice and peace.

ACKNOWLEDGEMENTS

I would like to thank Dr. Anu Bourgeois for her guidance and support during this research. I also would like to thank to the members of the committee, Dr. Raheem A. Beyah and Dr. Yingshu Li for their helpful comments and suggestions.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVATIONS

ACELP	Algebraic code excited linear prediction.
ADPCM	Adaptive differential pulse code modulation.
ATA	Analog telephone adaptor.
CBQ	Class base queuing.
CBR	Constant bit rate.
CELP	Code exited linear prediction.
CQ	Conversational quality.
CS-ACELP	Conjugate structure algebraic code exited linear prediction.
CSMA/CA	Carries sense multiple access with collision avoidance.
CSRC	Contributing source.
DVI4	Digital visual interface.
EDCA	Enhanced distribution channel access.
FEC	Forward error correction.
FIFO	First in first out.
GSM	Global system for mobile communications.
H.323	A signaling protocol defined by the International telecom union.
HCF	Hybrid coordinated function.
IEEE	Institute of Electrical and Electronics Engineer.
IETF	Internet engineering task force.
IFG	Inter frame gap.
IPG	Inter packet gap.

LD-CELP	Low delay code exited linear prediction.
LQ	Listening quality.
LPC	Linear predictive coding.
MGCP	Media gateway control protocol.
MOS	Mean opinion score.
MP-MLQ	Multi-pulse maximum likelihood quantization
MPC-MLQ	Multi-pulse linear predicting code with maximum likelihood quantization.
MPLS	Multi protocol label switching.
MAPDV	Mean absolute packet delay variation.
MPPDV	Mean packet to packet delay variation.
NPLC	No packet loss concealment.
PCMA	Pulse code modulation aLaw.
PCMU	Pulse code modulation uLaw.
PESQ	Perceptual evaluation of speech quality.
PLC	Packet loss concealment.
POTS	Plain old telephone service.
PSQM	Perceptual speech quality measurement.
PSTN	Public switch telephone network.
QOS	Quality of service.
RTCP	Real time control protocol.
RTCP-XR	Real time control protocol extended report.
RTP	Real time protocol.
RTT	Round trip time.

RUDP	Reliable user datagram protocol.
SCCP	Skinny client control protocol.
SIP	Session initiation protocol.
SSRC	Synchronization source.
TCP	Transmission control protocol.
UDP	User datagram protocol.
VAD	Voice activity detection.
VoIP	Voice over IP.

CHAPTER 1 – INTRODUCTION

Voice over IP (VoIP), the transmission of voice over the Internet, particularly the IP protocol, has gained popularity over the recent years. Businesses and institutions are transforming their current phone infrastructure from Plain Old Telephone Service (POTS) to VoIP. There is a notorious advantage of transmitting voice using the IP protocol and that is IP networks are low in cost. In fact, when a long distance call is placed over IP networks, it eliminates access charges to voice carriers.

VoIP also allows the integration of voice and data over the same channel. This translates to a new generation of applications. Voice mail can now easily be integrated into email, virtual conference rooms are being placed around the world, and services, such as caller ID and call forwarding, can be easily implemented in a packet switched network instead of the traditional circuit switched network. In the near future, we will see an overhaul of new services, such as wireless VoIP. In fact, VoIP is shaping the future of communications.

Over the last few years there has been remarkable progress in the field of VoIP. The Telecom industry has concentrated on developing new voice codecs that perform better under conditions of packet loss, and consequently increase the quality of VoIP calls to a certain extent. VoIP has also benefited from improvements in digital signal processing. Chips are being specifically designed to run certain type of voice codec algorithms. At the same time, there have been other efforts to provide a better QoS to VoIP, such as class based queuing (CBQ) that differentiate traffic based on IP source addresses, and multi protocol layer switching (MPLS) that provides fast forwarding of

packets at the router level. In addition, switches transmit at speeds of 1 and 10 Gbps. However, the quality of VoIP calls under impairment conditions such as delay, jitter, and packet loss still remains a major problem that needs to be address for next generation of VoIP services.

Through this research we are particularly interested in analyzing the effects that these impairments have in the quality of VoIP calls and different approaches to solve this problem. To measure the mean opinion score (MOS) of VoIP calls in the event of long absolute delays and packet loss we develop an application in JavaScript based on the E-Model [4]. On the other hand, to measure the MOS in the event of jitter buffer discards due to high variable inter-packet delays we utilize an intrusive technique, such as perceptual evaluation of speech quality (PESQ) [6]. Our contribution is to make a comparative analysis of the MOS for a VoIP call when the receiver implements a fixed and an adaptive jitter buffers respectively. We also analyze the performance of IP phones from different vendors under the event of constant delay, random delay, Gaussian delay with packet reordering, and Gaussian delay without packet reordering.

Finally, we analyze the limitations of introducing forward error correction (FEC) Piggybacking and Reed Solomon codes in VoIP mainly because these two techniques have been proposed over the past years to provide a better quality to VoIP calls under the event of packet loss [5][39].

The rest of this thesis is organized as follows. Chapter 2 presents background information in the field of VoIP. Chapter 3 is dedicated to analyze the factors that affect the quality of VoIP calls. Chapter 4 analyzes the impact and feasibility of introducing forward error correction (FEC) piggybacking and Reed Solomon codes in VoIP. Chapter

5 describes in detail the experiments set up during this research and a minor analysis of the results. Finally, in Chapter 6 we conclude that the effectiveness of forward error correction for VoIP depends on the one-way delay, nominal jitter buffer size, codec implemented, transmission cycle of the RTP stream, and congestion in a computer network.

CHAPTER 2 – BACKGROUND

This chapter presents fundamental knowledge in the field of voice over IP (VoIP) necessary to understand the coding, protocols, and innovative techniques used to transmit real time audio. We also present methods to measure the quality of VoIP calls and limitations of transmitting VoIP calls over a packet switch network such as the Internet.

2.1 Codecs

Codecs are an integral part in the development of VoIP, because they are responsible for the conversion of audio speech signal into encoded digital data. A codec is a software program that is able to encode a waveform generated at the sender side into a string of bits that when decoded at the remote end would be as similar as the original waveform.

Voice codecs break the original waveform into small chunks of data, and then every chunk is compressed using a specialized algorithm. Some codecs operate at rates of 64 kbps, while others operate at lower rates. Codecs that operate at high data rates tend to consume more bandwidth. However, they are able to better reassemble the original waveform at the remote end. On the other hand, codes that operate at lower data rates tend to consume less bandwidth, but the reassembled waveform generated at the remote end is not as similar as the original waveform. Therefore, the quality of a VoIP call is degraded just for the fact of using a low data rate codec. According to Cherry, the characteristics on the voice quality are significantly affected by the voice codec [11].

It is not always convenient to implement VoIP with codecs that operate at high data rates, because in the event of network congestion, it would make more sense to implement a low data rate codec. Table 2.1 describes the audio encoding sampling rates and bits per sample of most common encoding techniques, where N/A stands for not applicable, and var stands for variable [2].

Table 2.1. [2]. Properties of Audio Encoding

Encoding name	**Sample/frame**	**Bits/sample**	**Sampling rate**	**Default Ms/frame**	**Default Ms/packet**
DVI4	Sample	4	Var.	20	
G722	Sample	8	16,000	20	
G723	Frame	N/A	8,000	30	30
G726-40	Sample	5	8,000	20	
G726-32	Sample	4	8,000	20	
G726-24	Sample	3	8,000	20	
G726-16	Sample	2	8,000	20	
G728	Frame	N/A	8,000	2.5	20
G729	Frame	N/A	8,000	10	20
G729D	Frame	N/A	8,000	10	20
G729E	Frame	N/A	8,000	10	20
GSM	Frame	N/A	8,000	20	20
GSM-EFR	Frame	N/A	8,000	20	20
L8	Sample	8	Var.	20	
L16	Sample	16	Var.	20	
LPC	Frame	N/A	8,000	20	20
MPA	Frame	N/A	Var.	Var.	
PCMA	Sample	8	Var.	20	
PCMU	Sample	8	Var.	20	
QCELP	Frame	N/A	8,000	20	20
VDVI	Sample	Var.	Var.	20	

The most common codecs for VoIP are described in Table 2.2. Some codecs use advanced algorithms and techniques to model human vocal tracks. So, in the event of packet loss, the lost information can be reassembled based on information contained in

neighboring real time protocol (RTP) packets. To clearly explain this idea, assume that user A sends 5 RTP packets with modeled vocal track payload to user B. Also assume that only the first 3 packets and the 5th packet arrived to user B. Then user B can infer how packet number 4 might look like based on payload information from packets 1 to 3 and 5. Therefore, the loss of one packet in a stream of packets is almost imperceptible to the recipient, if the codec implemented has some sort of intelligence.

Table 2.2. VoIP common codecs

Codec	**Coding Method**	**Bit rate (kbps)**	**MOS-ITU**
G.711A (no PLC)	PCMA	64	4.40
G.711A PLC	PCMA	64	4.40
G.711U (no PLC)	PCMU	64	4.40
G.711U PLC	PCMU	64	4.40
G.721	ADPCM	32	4.23
G.723.1	MP-MLQ	6.3	3.95
G.723.1	ACELP	5.3	3.78
G.726	ADPCM	16	2.95
G.726	ADPCM	24	3.51
G.726	ADPCM	32	4.23
G.726	ADPCM	40	4.36
G.727	ADPCM	16	2.84
G.727	ADPCM	24	3.83
G.729	LD-CELP	8	3.92
G.729A	CS-ACELP	8	3.7

2.2 Voice activity detection and silence suppression

It is well known that human conversations over the phone consist of long periods of silence. Figure 2.1 presents this phenomenon in more detail. This graph was obtained during a VoIP call between two parties, a caller and a callee.

From Figure 2.1, it can be inferred that human conversations over the phone are composed of talk spurt and silent periods. A codec can benefit from this property and does not need to send packets into the network representing periods of silence. Thus, saving network bandwidth.

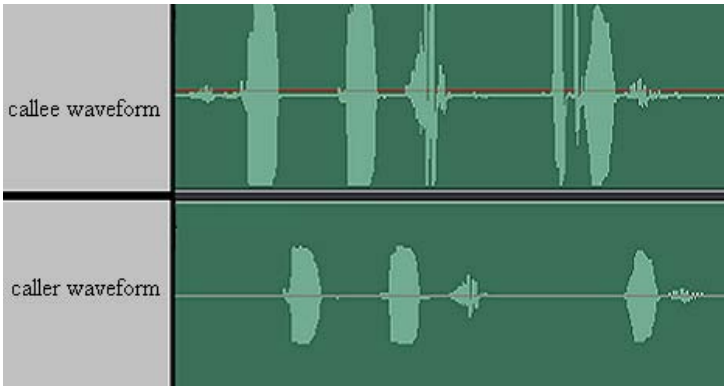

Figure 2.1. Speech characteristic of a call between two participants

Silence suppression implements a voice activity detection (VAD) mechanism that only generates RTP packets when the sender produces a voice signal. Therefore, no RTP packets are generated when the sender remains in silence or the surrounding environment noise is under an acceptable margin level.

Some codecs, such as G.711A with the voice activity detection feature turned-off, do not save bandwidth over periods of silence, because even the silence is sampled, packetized, and sent into the network.

An analysis made on the speech data rate of VoIP calls in the Thai language, found that the mean distribution of data rate conversational speech is 27 packets per second [12]. The codec utilized was G.729A in silence suppression mode, which operates at data rate of 8 kbps, with a transmission cycle of 20 msec and payload size of 20 bytes.

The same analysis found that for other languages, such as English, the mean distribution of data rate conversational speech is 19 packets per second [12]. This reinforces the theory that the conversational data rate depends on the language spoken.

Silence suppression saves network bandwidth because a non-silence suppression codec operating at a transmission cycle of 20 msec will be sending 50 packets per second into the network all the time. Therefore, silence suppression reduces congestion.

Notice that a comfort noise silence is generated at remote ends to simulate the low level noise that is similar as the one we are used to hear from the Plain Old Telephony Service (POTS). According to Chong and Matthews, VoIP handles all the features and enhances those previously supported in POTS [13].

2.3 Real time protocol

Real time protocol (RTP) was defined to provide an appropriate distribution of real time streaming of voice and video in computer networks. The TCP protocol is not appropriate for VoIP because there is too much delay associated with retransmissions. In turn, the UDP protocol is used at the transport layer. However, UDP has some weaknesses in that it does not ensure that packets will be delivered in the proper order.

Figure 2.2 describes the header of the user datagram protocol (UDP). Note that UDP consists of 4 fields only, and none of these fields contains a sequence number as in TCP. This can lead to problems when transmitting audio packets, because, if packets arrive at the destination out of order, there is no mechanism to re-order packets. The lack of a sequence number can also lead to other major problems because a receiver is not be able to determine if an audio packet got lost in the network

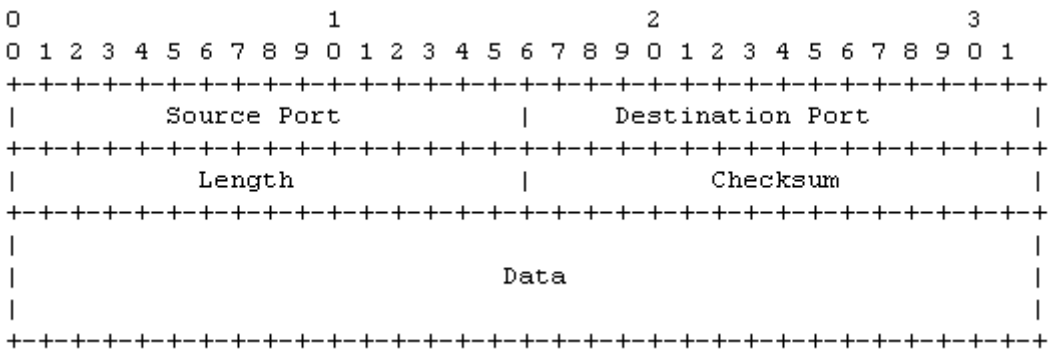

Figure 2.2. Headers of the user datagram protocol

The RTP protocol was designed to work in conjunction with UDP at the transport layer to ensure a proper delivery of packets with real time characteristics.

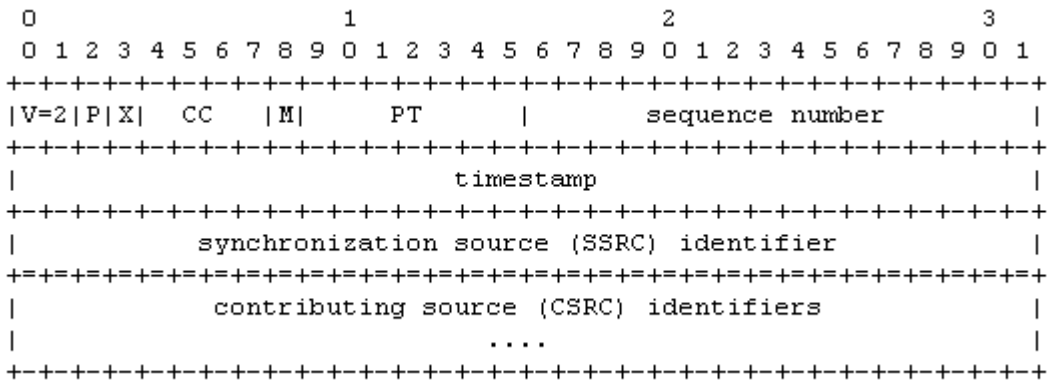

Figure 2.3. [1]. Headers of the real time protocol

Figure 2.3 describes the headers of the RTP protocol according to RFC 3550 [1]. Note that with RTP, every packet has now a sequence number and a timestamp. The

sequence number allows the receiver to order packets. The timestamp assists the receiver to identify the time at which voice packets were generated at the sender side. The headers of the RTP protocol are described below.

V - Refers to the version of the RTP protocol, currently version two.

P - Padding. If this flag is set to 1 then the RTP header contains one or more octet headers that do not belong to the payload. Some encryption algorithms need padding [1].

X - Extension. If this flag is set to 1 then the header must be follow by exactly 1 header extension.

CC - This field contains the number of CSRC identifiers that will be shown in the CSRC field. Since this field contains only 4 bits, the maximum number of participants that can be handled by a mixer is 15. Note that in the case that a mixer is not utilized then this value is equal to 0.

M - Marker. According to Schulzrinne, The marker is intended to allow significant events, such as frame boundaries to be marked in a packet stream [1].

PT - Payload type identification. It represents the format of payload data. Table 2.3 describes a set of mappings from most common voice codecs to payload types. It was extract from RFC 3551 and it represents the mappings established at the time the specification was written, where dyn stands for dynamic [2].

One interesting feature of VoIP is that participants can change codecs during an ongoing session by altering the payload type (PT). This was specifically designed to switch codecs in the event of a low bandwidth or congestion in a computer network [1]. Note that the PT field is only 7 bits long. Therefore, it is only possible to define up to 127 mappings from codecs to payload types. Moreover, it is predictable that soon this field

will run out of bits to map new codecs. So, additional payload types can be defined dynamically through non-RTP means, such as signaling [1].

Table 2.3. [2]. Payload types for audio encoding

PT	Encoding name	Media type	Clock rate (Hz)	Channels
0	PCMU	A	8,000	1
1	Reserved	A		
2	Reserved	A		
3	GSM	A	8,000	1
4	G723	A	8,000	1
5	DVI4	A	8,000	1
6	DVI4	A	16,000	1
7	LPC	A	8,000	1
8	PCMA	A	8,000	1
9	G722	A	8,000	1
10	L16	A	44,100	2
11	L16	A	44,100	1
12	QCELP	A	8,000	1
13	CN	A	8,000	1
14	MPA	A	90,000	(see text)
15	G728	A	8,000	1
16	DVI4	A	11,025	1
17	DVI4	A	22,050	1
18	G729	A	8,000	1
19	Reserved	A	8,000	1
20	Unassigned	A		
21	Unassigned	A		
22	Unassigned	A		
23	Unassigned	A		
Dyn	G726-40	A	8,000	1
Dyn	G726-32	A	8,000	1
Dyn	G726-24	A	8,000	1
Dyn	G726-16	A	8,000	1
Dyn	G729D	A	8,000	1
Dyn	G729E	A	8,000	1
Dyn	GSM-EFR	A	8,000	1

Sequence number - Refers to the sequence number of each RTP packet and it increases at increments of one. The sequence number is initially randomly generated to avoid a

plaintext attack in the event that encryption is needed. The sequence number helps the receiver to reorder out of order packets. It also allows detecting loss of packets. Note that the receiver can notify to the sender the rate of packet loss via a real time control protocol (RTCP) message. So, the sender can adjust the rate of the codec to a lower value, because it is assume that congestion is the main cause of the problem. This is known as adaptive encoding [1].

Timestamp - Refers to the time at which the first field of the RTP packet was generated. This time is not related to the system clock reading. This time depends on the sampling rate of the codec. For example, G.711 PCMA samples the medium 8000 times per second, and every sample contains 8 bits. For a transmission cycle of 20 msec only 50 packets can be sent into the network per second. Therefore, 8000 samples have to be distributed in 50 packets per second. This means that every RTP packet contains data worth of 160 samples. In conclusion, the timestamp field of every RTP packet for this particular example increases at increments of 160. The initial RTP timestamp value is randomly generated to avoid a plain-text attack in the event that encryption is needed [1].

Synchronization Source (SSRC) - It is a 32 bit numeric value and it represents a unique id that identifies all the RTP packets that belong to the same session. The receiver at the remote end reads this field to identify all the packets that belong to the same time and sequence number space [1]. This value is randomly generated. However, the algorithm that generates this value prevents duplication of SSRC between participants that want to initiate a new RTP session.

Contributing source (CSRC) - It represents a list of the SSRC identifiers from sources that contributed to a new flow of RTP packets produce by a mixer [1]. The CC field

supplies the number of SSRC identifiers in the list. A mixer is an intermediate system that combines the received RTP packets from various participants and produces a new RTP packet, possibly with a different voice codec, and a new timestamp. The CSRC field only appears in the header of a RTP packet in the presence of a mixer [1].

Note that with a timestamp and sequence number a receiver is able to reconstruct the timing necessary for playback. The RTP protocol specification is suitable for audio and video. As a matter of fact, for video streaming, two RTP sessions are established independently with different SSRC identifiers, one is used for video and another for audio. One of the main reasons of separating audio and video is because in a conference participants have the ability to control which medium they would like to receive [1]. In fact, The RTP protocol is suitable for unicast and multicast sessions. Moreover, RTP does not guarantee delivery of packets or quality of service [1].

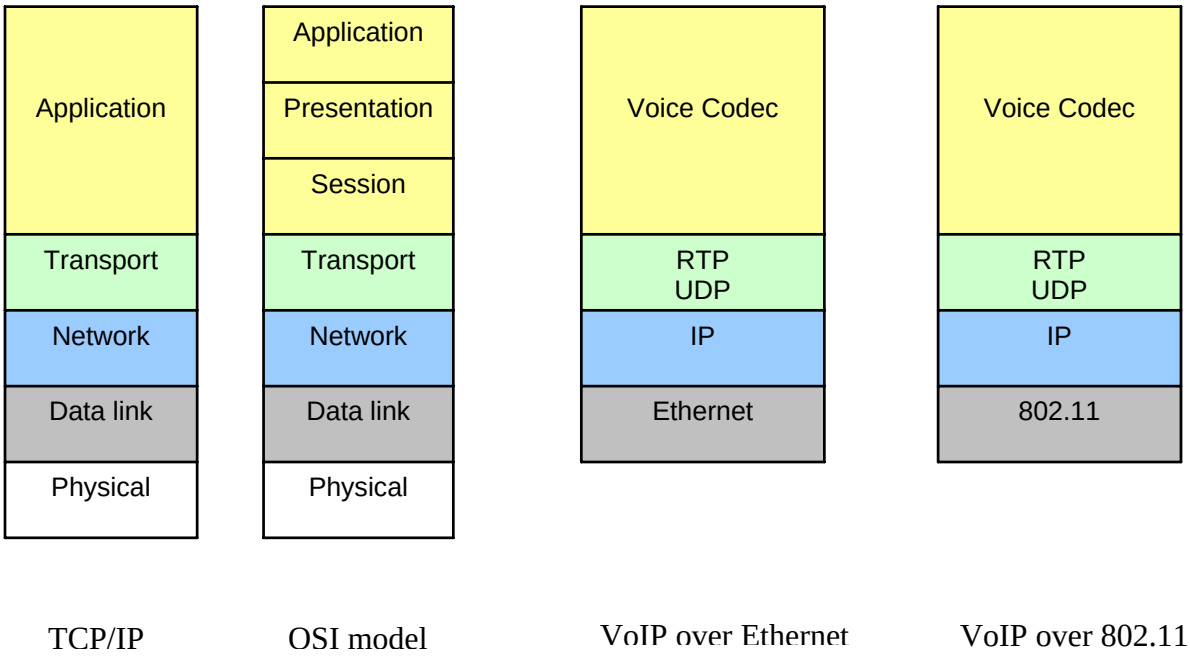

Figure 2.4. Real time protocol in the TCP/IP protocol stack

2.4 Real time control protocol

The real time control protocol (RTCP) was specifically design to provide reception quality feedback of RTP data distribution to all the participants in a session. Note that RTCP packets do not carry payload audio.

The feedback is performed by the sender (SR) and receiver reports (RR), which look similar. As a matter of fact, the sender report is only use to provide transmission and reception statistics from participants that are active senders. On the other hand, the receiver report is used to provide reception statistics from participants that are not active senders [1].

This feedback is useful to the sender because once interpreting this report decisions can be made, such as transmit data at lower rates using a high compression codec in order to compensate for congestion or another algorithm if it provides a better performance. Note that not always reducing to a low bit rate codec improves the quality of a call because in the event of transient packets lost, it makes more sense to implement other mechanism such as forward error correction (FEC). Moreover, these feedbacks can be use by network managers to determine if the network problems are local or regional [1].

One of the most important fields in the RTCP protocol are fraction lost, cumulative number of packet loss, and interarrival jitter. Fraction lost refers to the fraction of RTP packets from source SSRC lost since the previous SR or RR packet was sent. It is defined by the number of packet lost divided by the number of packets expected. If the loss is negative due to duplicate packets then fraction lost is automatically set to 0 [1]. Cumulative number of packet loss refers to the total number of

RTP packets from source SSRC lost since the beginning of the session. It is defined by the number of packet expected less the number of packet received. Note that packets received include the ones that are duplicate or late [1].

RTCP packets are sent at periodic intervals. Note that a good implementation of RTCP should not saturate the network with packets. As a matter of fact, participants sends only one RTCP packet per report interval. Moreover, the recommended value for a minimum fixed interval for sending RTCP packets is 5 seconds [1]. In addition, the RTCP packet interval also depends on the number of participants present in the session.

The RTCP protocol has a weakness in that it reports packets arrived even in the event of late packets. An enhanced version of RTCP is real time control protocol extended reports (RTCP-XR) that reports to the sender the lost of packets due to jitter buffer discards [9].

2.5 Jitter and jitter buffers

Jitter is defined as the variation in interpacket delay. To explain this idea in more detail we will calculate the jitter between two RTP packets. Assume that a sender generates the first RTP packet and stamps it with the time at which the packet was generated. This value will be written in the timestamp field of the RTP packet defined in Figure 2.3. Additionally, we will assume that the codec samples the medium 8000 times per second, and that the transmission cycle is 20 msec. Therefore, the timestamp increases at increments of 160. We will also assume that the timestamp field of the first packet is equal to 0. Therefore, after 20 msec, the sender will generate the next RTP packet. The timestamp field of this packet is of course equal to 160.

At the remote end the receiver collects the first RTP packet and establishes this time as a reference for the next incoming packets. Since the receiver knows that the sampling rate of the codec negotiated is 8000 samples per second, and that the difference of consecutive RTP packets timestamp fields is 160. It can be inferred that every packet should arrived at the receiver at 20 msec interval of the receiver's clock.

For this particular example it will be assume that the first packet arrives at 0 msec of receiver's clock and that the second packet arrives at 25 msec of the receiver's clock due to a queuing delay at intermediate routers. Therefore, we can estimate that the jitter is equal to 5 msec. This value is obtained as follows. Jitter of consecutive packets is equal to the difference of arrival time and timestamp for consecutive RTP packets. Note that the arrival time of RTP packets is measured by the receiver's clock.

$$Jitter\ of\ consecutive\ \ packets = (25\,msec - 0\,msec) - (20\,msec - 0\,msec) = 5\,msec$$

We can generalize this idea and write the following formula to calculate the jitter between any pair of adjacent RTP packets [1]. If S_i refers to the time at which a packet was time stamped and R_i refers to the time at which a packet arrived, we can state.

$$Jitter\ of\ consecutive\ packets = (R_i - R_{i-1}) - (S_i - S_{i-1}) = (R_i - S_i) - (R_{i-1} - S_{i-1})$$

The interarrival jitter is calculated from RTP packets with the same synchronization source (SSRC) identifier. Recall that the SSRC is the unique identifier that classifies RTP packets from a particular session.

It is known that there are two kinds of jitter. The first one is known as constant jitter, it happens when consecutive samples of jitter are relatively similar. The second one is known as transient jitter, it happens when there is a high variance in the jitter value between consecutive RTP packets. In fact, transient jitter is more detrimental to VoIP than constant jitter.

There are many statistical methods to measure the overall jitter of a VoIP call. MPPDV is defined as the mean of packet-to-packet delay variation. Assume that Db and Da is the delay between two consecutive RTP packets. Then the variation is defined as abs (Db – Da). Therefore, MPPDV = mean (abs ($D_i – D_{i-1}$). Note that when the standard deviation of the mean distribution is small it can be inferred the network is introducing a constant jitter. On the other hand, if the standard deviation is large then the network is introducing high transient jitters.

A jitter buffer is an essential component in VoIP. Its goal is to cancel the effect of variable interpacket delay that is always induced in computer networks. To properly describe the behavior of a jitter buffer, we will examine the following case.

A VoIP call is established between a sender and a receiver with a G.711 codec operating at a transmission cycle of 20 msec. Therefore, only 50 packets can be sent into the network per second, and every packet can only contain 20 msec of payload data. If these RTP packets are sent on a computer network with a optimal round trip time “RTT less than 1 msec”, even in the best conditions we can not deliver the information immediately to the receiver because while the last 20 msec of audio from the first packet is being rendered to the receiver, the next incoming packet might not be at the receiver yet. In fact, this packet can still be in the queue of a router. Consequently the receiver

might not see every packet arriving at a constant interval of 20 msec. Therefore, this can lead to starvation.

The jitter buffer, which is implemented at the receiver side, has to avoid the starvation state, so it holds the first packet in its buffers for a small amount of time until the next packet arrives and it is safe to start delivering the first packet [36]. This is known as nominal jitter buffer. Jitter buffers introduce an additional delay in the communication process. In this thesis we only perform experiments on nominal jitter buffer sizes of 40 and 60 msec.

An adaptive jitter buffer is an intelligent process, because it implements several strategies to render voice to the receiver. When network conditions are optimal, meaning that the variation in inter-packet delay is proximal to 0 msec, it adjusts itself to a minimum value to reduce latency. On the other hand, when network conditions are far from optimal, meaning that there is high transient jitter and packet loss, it adjusts itself to a higher value. This of course is necessary to deliver a better signal but the trade-off is more latency.

If the jitter buffer size is too small this can lead to two kinds of problems. First, if packets arrive too late to the receiver the effect will be packet discard by the jitter buffer. Secondly, if a burst of packets that arrive at the receiver is greater than the jitter buffer size then the effect is again packet discards. Notice that packet discard is equivalent to packet loss. On the other hand, if the jitter buffer size is too large this introduces an unnecessary delay that can lead to conversational degradation [9][36].

Without the implementation of a jitter buffer, a VoIP call will be only a broken audio signal. Moreover, the jitter buffer ensures that packets are delivered in correct

order. To conclude, default values for the maximum jitter buffer size are 80 to 120 msec with nominal jitter buffer sizes of 40 to 60 msec.

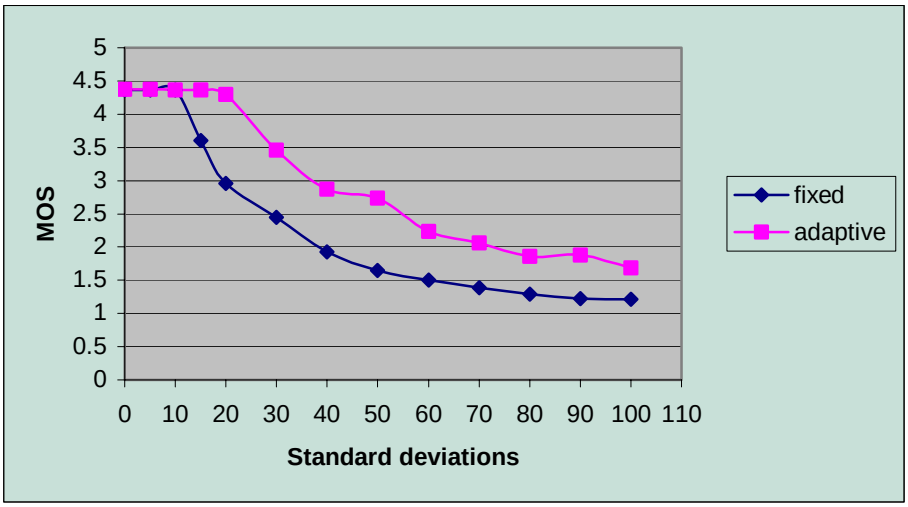

Figure 2.5. Comparative analysis of fixed vs. adaptive jitter buffer

Figure 2.5 represents a comparative analysis of the mean opinion score for a VoIP call when the receiver utilizes a fixed and an adaptive jitter buffer respectively. The mean opinion score was measured with perceptual evaluation of speech quality (PESQ). Refer to Section 5.4 and 5.5 for a detail explanation of the experiments setup to obtain these results.

2.6 Mean opinion score

The mean opinion score (MOS) is an average mark that is given by a panel of auditors to recorded samples. As a base line for this comparison, toll calls have a value of

4.2. The MOS is a value that can differ from auditor to auditor and has the interpretations as described in Table 2.4.

Table 2.4 [23]. Mean Opinion Score

Rating	**Speech**	**Description**
5	Excellent	Imperceptible level of distortion
4	Good	Just perceptible level of distortion
3	Fair	Slightly annoying
2	Poor	Annoying but not objectionable
1	Unsatisfactory	Very annoying

According to Miglani [23], Table 2.4 represents the mean opinion score. The MOS ranges from one to five. One means unsatisfactory, and five means excellent. There are three methods to measure the MOS.

Audit panel - Humans listen to a set of pre-recorded samples and assign a score to each sample. This score can be in the range from one to five. Note that the value submitted by every auditor might be different, because this is a subjective value. In fact, the MOS depends on test conditions and participants [32].

Intrusive techniques - A pre-recorded audio file is sent over the network using the RTP protocol with some voice codec. The received audio file is compared against the original, and an estimation of the MOS is given. It is known that perceptual evaluation of speech quality (PESQ) has a high degree of accuracy and the evaluated MOS is proximal to a human perception [35]. Moreover, PESQ implements a fast Fourier algorithm in order to compare audio signals [31]. According to Clark and Holthaus, PESQ is inefficient when calculating the MOS for hundreds of calls traversing a network segment because of the

complexity of signal comparison that involves the fast Fourier algorithm. In fact, the Fast Fourier algorithm is a high computing process [32][33].

There are some disadvantages of using PESQ because in the event of constant delay in a computer network, 500 msec for example, the value reported by PESQ is optimal. The PESQ value should not be optimal under this condition because communications that have a delay greater than 150 msec are considered a one-way communication only [32][35]. In addition, intrusive tests generate additional network traffic that should be used for real calls. Moreover, PESQ misses the effect of determining quality metrics in the event of minimum packet loss because if the codec can efficiently recover data under this event, the value of PESQ is optimal. Note that this value should not be optimal because in real time streaming of audio, there should not be packet loss in the first place [32]. Therefore, PESQ should be used under certain circumstances only.

Non-intrusive techniques - Non-intrusive techniques came into the scene of VoIP call quality monitoring because of the inconvenience of the subjective tests, and the waste of resources consumed by intrusive techniques [32]. Non-intrusive techniques are able to determine the quality of VoIP calls based on methods such as the E-Model which is a transmission rating model define by the ITU [4]. The main output of the E-Model is a scalar rating of transmission quality known as “Rating Factor” R, which can be map into an estimate of the mean opinion score [4].

The E-Model takes in considerations impairment factors generated by today’s digital processing devices. Moreover, the E-Model has been enhanced through many revisions that better take in considerations effects, such as: room noise at the sender side,

random packet loss, low talker sidetone levels, and other parameters that affect conversational quality [4].

The rating factor R is calculated according to the following formula.

$$R = R_O - I_S - I_d - Ie\_eff + A$$

R_0 represents a basic signal-to-noise ratio, including noise sources, such as circuit noise and room noise. I_S represents a combination of all impairments which occur simultaneously with voice signal, such as objective loudness rating and distortion. Factor I_d represents the impairments caused by delay, such as, talker echo, listener echo, and long absolute delays. The equipment impairment factor Ie represents impairments caused by low bit rate codecs. The advantage factor A allows for compensation of impairment factors when there are other advantages to the user [4]. Therefore, it is possible to map the R factor into the estimate conversational MOS according to the following formulas.

$For \;\; R<0: \qquad MOS_{CQE} = 1$

$For \;\; 0<R<100: \;\; MOS_{CQE} = 1 + 0.035R + R(R-60)(100-R)7 * 10^{-6}$

$For \;\; R>100: \qquad MOS_{CQE} = 4.5$

Note that the output of E-Model has not been completely verified because of the large number of possible combinations of input parameters that affect the overall quality of calls. Therefore, modifications to the E-Model are currently under study [4]. Appendix B presents a JavaScript implementation of the E-Model.

Non-intrusive techniques can be implemented in the form of “embedded passive monitor agents” in a variety of probes, gateways and IP phones [32]. They implement novel solutions to obtain call quality metrics. One is a jitter buffer emulator that predicts the behavior of remote jitter buffers. Therefore, it is possible to predict if a packet is discarded due to excessive inter-packet delay. Note that jitter buffer discards are similar to packet loss [9].

2.7 Signaling

Signaling protocols play an important role in the process of setting up, maintaining and terminating a call. There are several signaling protocols for VoIP that range from media gateway control protocol (MGCP), session initiation protocol (SIP) the IETF standard, H.323 the ITU standard, and skinny client control protocol (SCCP) defined by Cisco. Currently SIP and H.323 are the two most common signaling protocols for VoIP. In fact, SIP is slightly more popular than H.323, because it is scalable and simple. However, it is known that for video applications, H.323 is preferred over SIP.

Note that an unreliable service, such as UDP is useful to carry signaling data, because it reduces the overhead of persistent TCP connections. Nonetheless, end-users can communicate with their respective gateways using either UDP or TCP messages for signaling and control messaging.

According to et al. [29], the H.323 standard is complex and incomplete, and there is no guarantee that devices will interoperate properly even if all of them are H.323 compliant. H.323 uses many ports in the call setup process. As time has passed, H.323

and SIP have learned from each other. Therefore, new versions of these signaling protocols are optimized.

According to the IETF, MGCP is suitable for the Master-slave architecture, and the SIP protocol that was develop later is suitable for the Peer-to-Peer architecture. SIP is an application layer control protocol for establishing and terminating sessions with one or more participants. In fact, SIP can invite parties to either unicast or multicast sessions [22].

2.8 Session Initiation protocol

Session Initiation protocol (SIP) is a signaling protocol for VoIP and it was designed by Dr. Henning Schulzrinne at Columbia University. SIP is a text-based lightweight protocol that remains simple and scalable. It can be used to create, modify, and terminate sessions with one or more participants. SIP can be used with any transport layer protocol TCP or UDP typically using port 5060 [10]. In this Section we describe the SIP registration, and call setup process between Cisco ATAs 188 and Ondo SIP proxy server. Note that calls can also be placed between SIP endpoints without the need of a SIP proxy server.

SIP registration - SIP is a lightweight protocol that has been optimized in order to facilitate the registration process in a fast and reliable manner with the least amount of packets exchanged between SIP clients and the SIP registrar.

1. The SIP client starts the registration process, by sending a register request to the IP address and port number of the SIP registrar server, this is known as SIP request method "REGISTER". This packet contains information, such as: IP address and port

number of the SIP client, phone number assigned to the SIP client, and registration expiration time.

2. If the SIP registrar server is available and listening in a predefined port, it will reply back as soon as possible to the SIP client with a SIP response method "100 Trying". The main purpose of this packet is to tell the SIP client that its registration request is in process. Notice that if the SIP client does not receive this message, it will try to resend a register request after a certain amount of time.

3. If the SIP registrar server successfully registers the SIP client, then a new message is sent back to the SIP client. This is known as a SIP response method "200 OK". Note that this method resembles in some way the responses of the HTTP protocol. Figure 2.6 represents a flow graph of the SIP registration process between a SIP client (192.168.10.2) and a SIP proxy server (192.168.10.20).

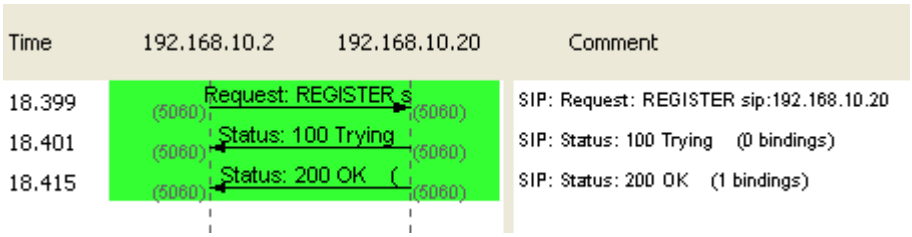

Figure 2.6. SIP Registration process

SIP call setup – To explain this process in more detail we will describe the scenario of Figure 2.7.

1. The caller (192.168.10.3) starts the call setup process by dialing the callee phone number. Therefore, the caller uses the SIP request method “INVITE” that has the following format destination_number@SIP_proxy_server_IP_address, and forwards this

packet to the SIP proxy server. Note that it is responsibility of the SIP proxy server to map the destination phone number into the callee IP address. Moreover, this packet also carries the phone number and IP address of the caller, and media attributes of the RTP session, such as prefer codecs.

2. If the SIP registrar server is available and listening in a predefined port, it will reply back immediately to the caller with a SIP response method "100 Trying". The main purpose of this packet is to tell the caller that it is processing the call setup request.

3. The SIP proxy server then tries to contact the callee that was already registered to the SIP proxy server. Therefore, mapping of phone number to an IP address is known in advance. The proxy server uses a SIP request method “INVITE” with the following format destination_number@callee_IP_address, and forwards this packet to the callee. Moreover, this packet also carries the phone number and IP address of the caller, and media attributes of the RTP session, such as prefer codec type.

4. If the callee is available and listening in a predefined port, it will reply back immediately to the SIP proxy server with a SIP response method "100 Trying". The main purpose of this packet is to tell the SIP proxy server that it is processing the call setup request.

5. If the callee accepts the call setup request, it sends to the proxy server a SIP response method known as "180 Ringing". Consequently, the callee starts ringing. Moreover, the SIP proxy acknowledges this message and forwards it to the caller.

6. The caller acknowledges this message and sends to the SIP proxy server a SIP request method known as provisional acknowledgement (PRACK). Moreover, the SIP proxy forwards the PRACK to the callee.

7. The callee acknowledges the PRACK message and responds to the SIP proxy a SIP response method known as “200 OK”. Moreover, the SIP proxy forwards this packet to the caller.

Once this is accomplished both phones will keep on a stationary state until the callee picks up the phone. If the callee answers then a RTP session will be activated in both directions. Note that SIP does not carry any voice data because it is responsibility of RTP. Therefore, the quality of a call does not depend on the signaling protocol. However, it has a direct effect in the call setup time. It can be clearly seen in Figure 2.7 that the call setup time is 251 msec.

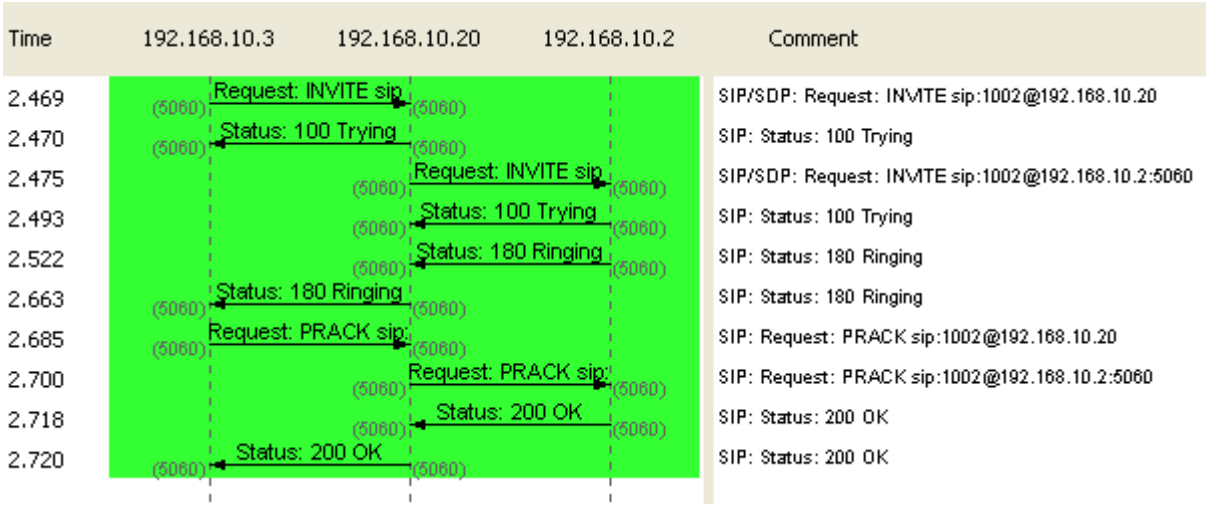

Figure 2.7. SIP call setup process

2.9 Constructing VoIP packets and throughput of one call with G.711 codec.

To clarify the process of constructing a sample VoIP packet, the codec G.711 will be used. The G.711 codec samples the medium 8000 times per second, and every sample

contains 8 bits of data. If we make a simple calculation we can infer that the data rate at which this protocol operates is 64,000 bps or 64 kbps.

The G.711 codec as well as other standard codecs can transmit data at different transmission cycles or intervals of time. For this particular example, we will use a transmission cycle of 20 msec. This means that in one second we can only transmit 50 VoIP packets. Consequently, if the data rate of the codec is 64 kbps or 8000 bytes/sec, then every packet can contain only 8000 bytes / 50 = 160 bytes of payload data.

To construct a VoIP packet we need to add the proper headers from all the layers of the TCP/IP stack. To the 160 bytes of codec payload data per packet we need to add the 12 bytes of RTP header [1]. We also need to aggregate the 8 bytes of UDP header and 20 bytes of IP header. Assuming that the packets will be sent over a wired connection, the Ethernet specification will add an overhead of 26 bytes including the preamble, headers, and trailers added at this layer. We also need to add the overhead introduced by the interframe gap. The interframe gap (IFG) or interpacket gap (IPG) is 12 bytes for the Ethernet specification at 100 Mbps, 1 Gbps and 10 Gbps. The IFG also consumes bandwidth and must be considered in our analysis. These headers sum up to a total of 78 bytes.

Consequently, for this particular example, every VoIP packet introduced into the network is composed of 160 bytes (payload) + 78 bytes (headers) = 238 bytes. This is only true when implementing the G.711 codec operating at 20 msec intervals over Ethernet networks. Therefore, since we transmit 50 packets per second, the actual throughput of a VoIP call using the G.711 codec without silence suppression operating at

20 msec packet interval is 238 bytes/packet * 50 packets/sec = 11900 bytes/sec or 95.2 kbps.

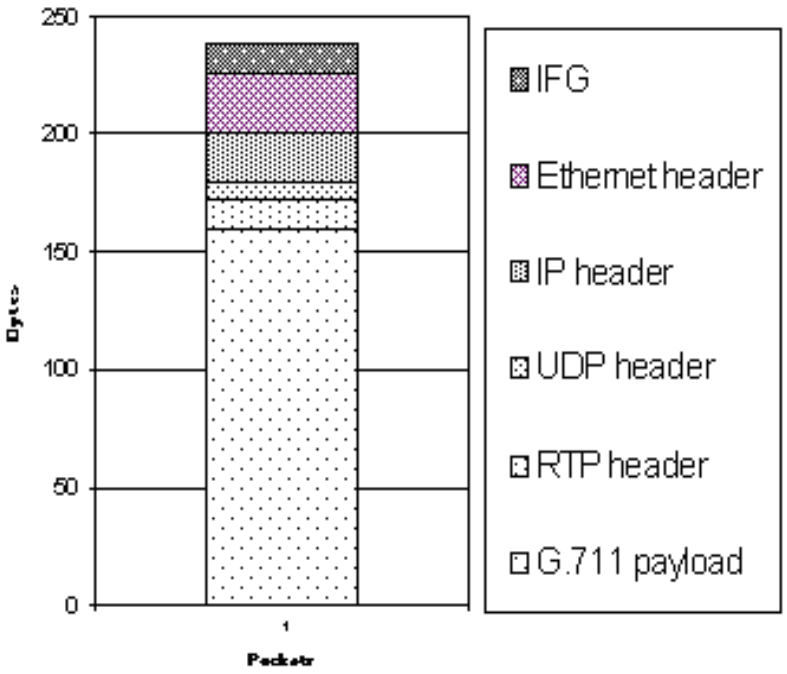

Figure 2.8. Proportion of payload and headers in G.711 at 20 msec packet intervals

2.10 A ratio of payload vs. packet length for codecs operating at 20 msec intervals.

From the analysis performed in Section 2.9, we can generalize and write the following equation to calculate the data payload per packet for all type of codecs operating at different transmission cycles. Notice that various codecs send voice at different data rates.

$$\text{Data payload per packet (bytes)} = \text{codec rate (kbps)} * \frac{1\,\text{byte}}{8\,\text{bits}} * \frac{1000}{1\,\text{k}} * \frac{\text{transmission cycle (msec)}}{1000\,\text{msec}}$$

Replacing the transmission cycle for 20 msec in the equation shown above, we obtain.

$$\text{Data payload per packet (bytes)} = \text{codec rate (kbps)} * \frac{1\,\text{byte}}{8\,\text{bits}} * \frac{1000}{1\,\text{k}} * \frac{20\,\text{msec}}{1000\,\text{msec}}$$

From Section 2.9, we know that the header size of VoIP packets on Ethernet networks is equal to 78 bytes. Therefore, we can infer.

$$\text{Packet length (bytes)} = \text{Data payload per packet (bytes)} + 78\,\text{bytes}$$

$$\text{Ratio of payload per packet vs. packet length} = \frac{\text{Data payload per packet (bytes)}}{\text{Packet length (bytes)}}$$

Table 2.5 can be constructed to calculate the ratio of payload vs. packet length in VoIP packets operating at different data rates with a transmission cycle of 20 msec. Note that this is only valid for packets sent over Ethernet networks.

The results obtained in Figure 2.9 have been published by OOUCHI, TAKENAGA, and SUGAWARA [3]. Our analysis confirms their result and presents a formulation in how to obtain the data. It can be clearly observed in Figure 2.9 that there is a disadvantage of transmitting at lower data rates because every packet that is sent into the network contains more headers than actual payload data. This overhead can be as high as 79.59% for codecs that transmit 20 bytes of payload per packet. For other codecs

that transmit 80 bytes of payload per packet, there is balance between the overhead and payload data per packet sent into the network. These results are only valid at 20 msec intervals.

Table 2.5. A ratio of payload vs. packet length

Codec rate (kbps)	**Payload per packet (bytes)**	**Packet length (bytes)**	**Ratio payload vs. packet length**
8	20	98	20.41
16	40	118	33.90
24	60	138	43.48
32	80	158	50.63
40	100	178	56.18
48	120	198	60.61
56	140	218	64.22
64	160	238	67.23

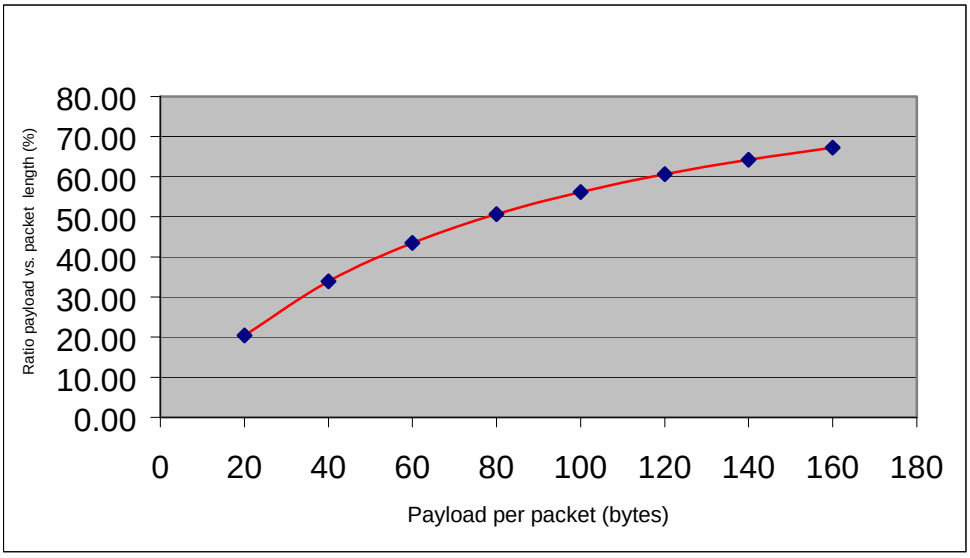

Figure 2.9. A ratio of payload vs. packet length.

2.11 Payload size per packet for G.711 and G.729 at different packet intervals.

The G.711 and G.729 codec transmit voice at data rates of 64 kbps and 8 kbps respectively. The payload size per VoIP packet varies drastically depending on the transmission cycle implemented. From the equations presented in Section 2.10, Table 2.6 can be constructed.

Table 2.6. Payload size per packet for G.711 and G.729 at different packet intervals

Transmission cycle (msec)	**G.711 (64 kbps)**	**G.729 (8 kbps)**
1	8	1
2	16	2
3	24	3
4	32	4
5	40	5
10	80	10
20	160	20
30	240	30
40	320	40
50	400	50
60	480	60
70	560	70
80	640	80
90	720	90
100	800	100
110	880	110

From the results presented in Table 2.6, we can infer that in the event of packet loss, it is convenient to send data at low transmission cycles, because the smaller the transmission cycle the smaller the amount of payload data that may get lost in the network. However, transmitting at low transmission cycles means that more packets have to be sent into the network, and that itself increases the probability to lose too many consecutive packets in the event of transient network problems. Moreover, in the analysis performed in Section 2.12, we observe that transmitting at low transmission cycles leads

to a huge waste in bandwidth resources. So, an optimal value for the transmission cycle is determined.

2.12 Bandwidth consumption of G.711 and G.729 at different packet intervals.

From the data acquired in Section 2.9 and 2.10, we can generalize and obtain the following equation to calculate the bandwidth consumption of a VoIP call using the G.711 and G.729 codecs operating at different transmission cycles over Ethernet networks. Note that for this analysis we are assuming no silence suppression for both codecs.

$$\text{Bandwidth(kbps)} = [\text{Data payload per packet(bytes)} + 78\text{bytes}] * \frac{8\,\text{bits}}{1\,\text{byte}} * \frac{1\text{k}}{1000} * \frac{1000\,\text{msec}}{\text{transmission cycle (msec)}}$$

Substituting transmission cycles with values in the range from 1 to 110 in the equation shown above, Table 2.7 can be constructed to find the actual bandwidth consumption of the G.711 codec operating at different transmission cycles.

Substituting the transmission cycles with values that range from 1 to 110, we can construct Table 2.8 to find the actual bandwidth consumption of the G.729 (8 kbps) codec operating at different transmission cycles. For this analysis we are assuming a constant bit rate for the G.729 codec. Note that G.729 codec may consume less bandwidth than what is stated in Table 2.8 because no packets are sent in the network if voice activity detection is implemented.

Table 2.7. Bandwidth consumption of the G.711 at different transmission cycles

Transmission cycle (msec)	**Payload size per packet (bytes)**	**Bandwidth consumption (kbps)**
1	8	688.0
2	16	376.0
3	24	272.0
4	32	220.0
5	40	188.80
10	80	126.40
20	160	95.20
30	240	84.80
40	320	79.60
50	400	76.48
60	480	74.40
70	560	72.91
80	640	71.80
90	720	70.93
100	800	70.24
110	880	69.67

Table 2.8. Bandwidth consumption of the G.729 at different transmission cycles

Transmission cycle (msec)	**Payload size per packet (bytes)**	**Bandwidth consumption (kbps)**
1	1	632.0
2	2	320.0
3	3	216.0
4	4	164.0
5	5	132.80
10	10	70.40
20	20	39.20
30	30	28.80
40	40	23.60
50	50	20.48
60	60	18.40
70	70	16.91
80	80	15.80
90	90	14.93
100	100	14.24
110	110	13.67

Combining the results from Tables 2.7 and 2.8, we obtain Figure 2.10.

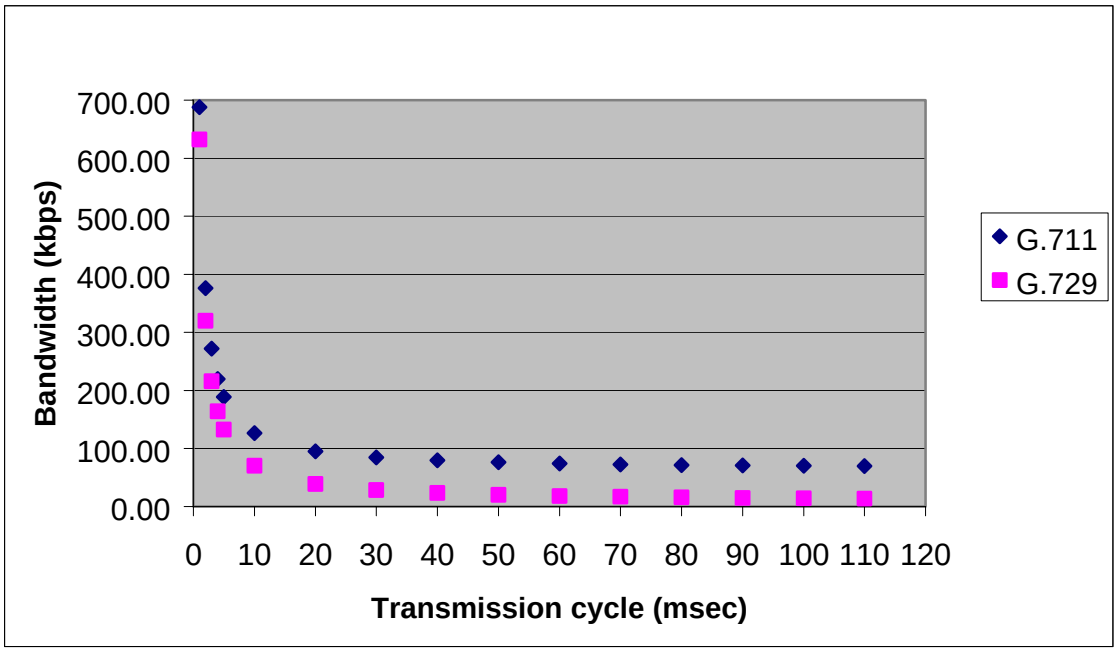

Figure 2.10. Bandwidth consumption of G.711 and G.729 at different packet intervals

The results obtained in Figure 2.10 have been published by OOUCHI, TAKENAGA, and SUGAWARA in [3]. Our analysis confirms their result and presents a formulation in how to obtain the data. It can be clearly seen in Figure 2.10 that the bandwidth consumption varies drastically depending on the transmission cycle. We can observe that both codecs G.711 and G.729 have great inefficiencies in bandwidth consumption operating at transmission cycles from 1 to 5 msec. However, at 20 msec interval the bandwidth consumption is not as worse as 5 or 10 msec and not much better than 30 or 40 msec. In fact, 20 msec is an industry standard for VoIP packet interval.

We can also notice in Figure 2.10 that the bandwidth consumed by a call using the G.729 codec with a transmission cycle of 20 msec is 39.20 kbps. However, the actual throughput of a call using G.729 is much less than 39.20 kbps, because G.729 in silence suppression mode does not send packets into the network over periods of silence. According to the observations made in Section 2.2, we can adjust this result, and state that the average throughput of a call using the G.729 codec operating at a transmission cycle of 20 msec in the English language is approximately 14.8 kbps. The calculation is described below.

Throughput = 98 (bytes/packet) * 19 (packets/sec) = 1862 bytes/sec

Throughput= 1862 (bytes/sec) * 8 (bits/byte)= 14896 bps

Throughput= 14,8 kbps

2.13 VoIP over wireless networks

From previous knowledge we know that communication channels over wireless networks are more susceptible to errors than their wired counterparts. Let us examine why sending voice packets over 802.11 wireless networks has many difficulties. For this particular example, assume a call using the G.711 codec operating at a transmission cycle of 20 msec.

First of all, RTP packets are generated and added the appropriate UDP headers at the transport layer. Then the proper IP headers are added at the network layer. The problem starts at the data link layer. Note that 802.11 is susceptible to packet loss due to the path fading effect, high collisions due to retransmissions, and interference from wireless devices operating at the same frequency. As a matter of fact, it has been reported

that 802.11b wireless devices are susceptible to interference in the presence of bluetooth devices. Moreover, when many devices were activated in proximity of the 802.11b cell, the quality of the call dropped to useless. An extensive analysis on the speech quality of VoIP in these environments is conducted for McKay and Masuda in [31].

If VoIP traffic is sent into the wireless medium in conjunction with other services that use the TCP protocol, the retransmission of lost packets of the TCP established connection could make it difficult for the RTP packet to fight for the medium every 20 msec. In fact when there is a collision, the RTP packet has to wait an exponential back off time before it tries to compete for the medium again. As a result, the RTP packets cannot be sent at a constant interval of 20 msec. This has a tremendous effect in the Mean opinion score of a VoIP call and even a jitter buffer working at its maximum size will suffer from starvation.

According to Garg and Kappes in [25], the maximum number of crystal clear calls in 802.11(a/b) networks is much lower than in Ethernet networks. In their analysis it was found out that 802.11b wireless networks can support a maximum of 3 to 12 calls when the G.711 codec is used operating at a transmission cycle of 20 msec. They also state that the execution of an additional call on the cell degrades the quality of all the calls. This number depends on the rate at which the 802.11b cell is operating: 1 Mbps, 2 Mbps, 5.5 Mbps or 11 Mbps. Moreover, an analysis on the limitations of the 802.11(a/b) distributed coordination function to support VoIP on wireless links as well as the upper bound number of simultaneous calls that can be placed on 802.11(a/b) networks with optimum performance is provided in [25].

Furthermore, according to Garg and Kappes, the maximum number of VoIP calls that can be handle by the IEEE 802.11b wireless specification is equal to 6 when the G.711a-Law codec is used operating at a transmission cycle of 10 msec [34]. Notice that RTP streams consume more bandwidth at 10 msec intervals rather than at 20 msec intervals.

The main problem of transmitting numerous calls in 802.11b wireless networks is directly related to the nature of 802.11 medium access protocols. Specifically, carrier sense multiple access with collision avoidance (CSMA/CA) mechanism, and the distributed coordination function (DCF) [34]. Note that in wireless networks collisions cannot be detected. So the MAC protocol was design to prevent collisions from taking place. Therefore, it was necessary to design a new protocol to overcome this problem, and as a result the protocol 802.11e was defined. With this protocol, RTP packets will not wait an exponential back off time to compete for the medium in a wireless environment. In addition, RTP/UDP packets are treated with priority over any TCP packet.

The MAC sub layer of 802.11e supports QoS. The MAC protocol is called Hybrid Coordination Function (HCF). The HCF is called hybrid because it combines a contention channel access mechanism, referred to as enhanced distributed channel access (EDCA). With EDCA a single MAC can have 4 queues that work independently, in parallel, and every queue has its own priority. A study on the advantages and limitations of IEEE 802.11e is given in [24].

There are other techniques that have been proposed to improve the number of simultaneous calls over 802.11 wireless networks, such as a multiplex multicast scheme. This technique has proven to be efficient to increase the VoIP capacity on wireless links

and decrease the probability of blocking. However, this idea does not have many supporters in the industry. In fact, the main disadvantage in using a multiplex multicast scheme is security, because all stations will receive packets with payload information from other active participants [26].

2.14 Security considerations for VoIP

VoIP is an application that offers low costs and great flexibility. However, it is vulnerable to high risks and threats. One of the most well known issues in securing VoIP is eavesdropping, which means that non-intended parties can listen to a call. Moreover, VoIP calls are also susceptible to denial of service attacks. In this Section we will describe these attacks and other complex cases as well.

VoIP eavesdropping can be easily implemented, using a network sniffer program, such as Ethereal [14]. In promiscuous mode, this software will capture all the RTP packets flowing in both directions. A script program that manipulates strings can be easily implemented to extract the payload of all sequential RTP packets. Note that the result of this operation will be one payload that contains all the information flowing in one direction. In addition, the information about the codec implemented in the call also travels in the RTP packets. According to RFC 3550, the payload type (PT) is a 7-bit value that identifies the RTP payload format [1]. Refer to the PT field in Figure 2.3. We can compare the PT field value from the RTP packet with the encoding at Table 2.3, and utilize a software application that decodes the payload to a .wav file. Moreover, this procedure can also be implemented over 802.11b wireless networks, due to the security weakness of the protocol [15].

VoIP calls are also susceptible to denial of service (DoS) attacks. As a matter of fact, VoIP has introduced new DoS attack in the world of computer networks. If a malicious procedure introduces a delay in the RTP flow that is a fraction of a second, then this can be considered as a successful attack [15].

The target of a VoIP attack is the information exchanged between two parties, the caller and the callee. It is known that a hacker can introduce speech in the voice line, without the sender and receiver even noticing, therefore, altering the information sent between the two parties in real time [16]. If there is an encryption mechanism in the communication, a hacker can find a key and implement a "Man in the middle attack" [18]. Moreover, the hacker can introduce data in the line at the same tone and pitch as the original sender.

Modem Hijacking is another example of VoIP attack. Estacion mentions in [17], "Users who have an standard dial up connection, once a malicious software is installed on their computers, a modem closes the internet connection when the user has been away of the computer for a considerable time, then the modem dials a long distance call. So the users have to pay for high phone bills. In the US and Canada people has filed complaints against FCC (Federal Communications Commission) and CRTC (Canadian Radio-Television Commission) respectively".

VoIP like other applications on the Internet is susceptible due to the operating system's vulnerability. It does not matter which operating system is being used (Windows, Linux, Mac or Solaris) [18]. It is known that there are complex attacks that target VoIP application servers, thus crashing or altering user's database information [16].

There are many ways to provide security to VoIP calls. One of the most important techniques is to encrypt the payload of RTP packets, in which the sender and the receiver exchange a key. However, there is a drawback added to the communication, and that is an additional delay has to be added to encrypt the data, and additional extra time to decrypt it. Therefore, encryption introduces jitter to a VoIP call.

IPSEC is a good technique to secure a VoIP call, because headers and payload of every RTP packet are encrypted. Additionally, IPSEC adds a new IP header so packets can be routed across the Internet. Note that with this process every packet increases in size. It is known that Skype provides a level of security to its customers because all its VoIP calls are encrypted with a 256 bit AES (Advance Encryption Standard) [19]. In the near future VoIP service providers will start to encrypt calls using this or another advanced algorithm. Moreover, the network appliances that we use to protect data like firewalls or VPN gateways introduce additional delay in the communication process, so security has to be addressed carefully [15][20].

Notice that even the SIP call setup delay that is the time between one of the participants creates a SIP INVITE message and receives a SIP ACK from the remote participant is incremented by and average of 0.7 sec when the call is send across a VPN tunnel running advance encryption standard (AES) or 3DES algorithm. This can be attributed to the key setup and authentication schemes in virtual private networks [20]. A comparative analysis of the effects of firewall and VPN techniques on the quality of a single call using the SIP signaling is conducted by Aire, Maharaj, and Linde in [20].

Quality of service is of fundamental importance to VoIP; this application is susceptible to high delays on a computer network. Note that a 150 msec delay can turn a

good voice call into a confused non-understandable sound [15]. Finally, the maximum permissible delay to fall within ITU recommendations is 150 msec [20].

2.15 Recommendations for improving the quality of VoIP calls

There are numerous techniques available to provide a better QoS to VoIP calls.

1. Ensure that the RTT from the sender to the receiver is less than 150 msec and has a stable pattern.

2. Minimize the latency or jitter induced by appliances, such as: router, firewall or VPN gateways. Also consider measuring the performance of these equipments under extreme conditions, where huge amount of data is crossing the network.

3. Use a hardphone rather than a softphone. Softphones tend to introduce high transient jitters when a user runs multiple programs in the same computer. Moreover, hardphones use specialized DSP chips to encode and decode certain type of VoIP codecs.

4. Ensure that the CPU utilization and network interfaces in a router are not over saturated, because under these conditions, routers are forced to drop packets, generate random inter-packet delays, and burst of packets.

5. Differentiate VoIP traffic from other kinds of traffic, and enforce on routers a queuing strategy different than first in first out (FIFO). It is known that class based queuing (CBQ) mechanism that differentiates traffic based on IP address, has great performance when VoIP traffic from known IP addresses is send across the network. Priority queuing is another good discipline to handle VoIP traffic.

6. Consider separate voice and data traffic using different IP address blocks.

7. Implement multi protocol label switching (MPLS) at the router level to provide fast forwarding of packets. This has a great impact in the QoS of VoIP because packets coming into one of the router interfaces do not have to wait to be processed at the router fabric. In fact, packets are fast forwarded to the next network segment as if they were generated in the same local area network.

8. Perform sanity interoperability tests to ensure compatibility between SIP, H.323 and public switch telephone network (PSTN) system. Notice that VoIP is a transition technology that needs to be backward compatible with the PSTN. Interoperability is known as one of the major problems to ensure reliability of the service. According to a study conducted by the Gartner Group in 2004 [29], 50% to 70% of all major projects fail due to software and interoperability problems. Interoperability problems are not only seen across VoIP equipments form different manufactures, but from H.323 and SIP devices from the same vendor as well. Furthermore, it is known that the most difficult task in VOIP is to provide interconnectivity to the POTS.

9. Disable unnecessary ports and services in SIP and H.323 gateways to decreased the possibility of unauthorized access and remote code execution [15].

Notice that even tough we mention all these aspects. There is yet another aspect that we cannot control and that is VoIP relies on the public Internet to send packets. According to a study conducted in 2004 by Chong and Mathews [28], the reliability of the POTS is 99.999%, this translates to five minutes of downtime per year, and the reliability of the public Internet is approximately 61%, this translates to 142 days of downtime per year.

CHAPTER 3 – PROBLEM DEFINITION

This Chapter discusses the factors that affect the quality of VoIP calls. After categorizing these factors, we will perform experiments that will further quantify their effects.

3.1 Factors that affect the quality of a VoIP call.

There are mainly four aspects that affect the quality of a VoIP call.

1. The nodal processing delay, meaning the time to digitize, compress and packetize voice data has to be kept the minimum as possible. The same delay is encountered at the remote end, because packets have to be de-packetize, uncompress, and the digital data has to be converted into audio. Some codecs introduce an additional nodal processing delay because in order to model human vocal tracks it is necessary to utilize more CPU cycles.

2. The accumulated delay from sender to receiver has to be less than 150 msec. According to Chong and Mathews in [28], the delay that is imperceptible to the human ear is 150 msec. Delays that are in the range of 150 to 250 msec are acceptable, but delays that are greater than 400 msec are unacceptable. Furthermore, ITU recommends that the maximum permissible delay for a real time communication is 150 msec.

3. The Jitter or variation in the delay between real time protocol (RTP) packets has to be under tolerant values, so the jitter buffer can manage this event. Note that when there is high jitter in a computer network, the result is packet discard by the jitter buffer,

which is equivalent to packet loss. In fact, even if packets arrive at the remote end, the jitter buffer will discard the packets that are out of an acceptable time frame [28].

Sections 5.2 and 5.3 show in detail the experiments setup to reproduce degradation of calls sent across analog telephone adaptors under the event of distinct form of delay such as fixed, uniform, Gaussian, and jitter. The results obtained during these experiments lets as concluded that fixed delay does not affect the listening quality of a call. However, it affects the conversational quality. Uniform delay is more detrimental to VoIP than Gaussian delay. Gaussian delay with packet reordering is more detrimental to VoIP than Gaussian delay without packet reordering. In addition, the quality of a VoIP call is different from vendor to vendor, this is mainly because vendors use different hardware, software, and jitter buffer algorithms.

Section 5.4 presents in detail the experiments setup to measure call quality degradation of a VoIP call sent across a generic analog telephone adaptor that uses a fixed jitter buffer under the event of Gaussian delay with packet reordering.

Section 5.5 present in detail the experiments setup to measure call quality degradation of a VoIP call sent across a generic analog telephone adaptor that uses an adaptive jitter buffer under the event of Gaussian delay with packet reordering.

4. Packet loss has to be under a maximum permissible limit. VoIP relies on RTP and UDP at the transport layer to send packets across networks and under conditions of packet loss, there is no retransmissions. This can be easily noticed if we observed the RTP and UDP header format. According to Walsh and Kuhn in [15], a 5% packet loss can make a call catastrophic. Previous research also states that the maximum tolerable packet loss is 3%. In fact, it can be clearly seen in Figure 3.1 that for 3% of random

packet loss probability the MOS of a call is really affected. However, some codecs have better performance than others at 3% packet loss.

If we assume a nominal jitter buffer size of 60 msec, VoIP should only be implemented in scenarios where the one-way delay is less than 80 msec, in order to be compliant with the 150 msec maximum permissible delay recommended by ITU. However, it has been demonstrated in experiments that VoIP can still be implemented in environments where RTT/2 is greater than 200 msec. Nevertheless, it must be a constant latency [27]. This is known as one-way communication. Refer to Sections 5.2 and 5.3 for an extensive analysis of the listening quality of a VoIP call under conditions of constant delay.

3.2 MOS under conditions of packet loss.

It is known that the event of packet loss in a computer network occurs in burst behaviors, meaning that there is a high probability that this event happens within packets being delivered around the same period of time. It has been observed through experimental traces in wide area networks that packet loss occurs as an isolated event or in burst patterns. Moreover, one of the main causes of packet loss in a computer network is congestion and transient network problems.

Figure 3.1 represents the mean opinion score of a VoIP call using different codecs for different values of random packet loss probability. In can be clearly seen that some codecs such as G.711 packet loss concealment (PLC) performs better than G.711 no PLC. Refer to Appendix B for the implementation of the E-Model in JavaScript. Figure 3.2 shows a snapshot of the JavaScript application implemented to calculate these results.

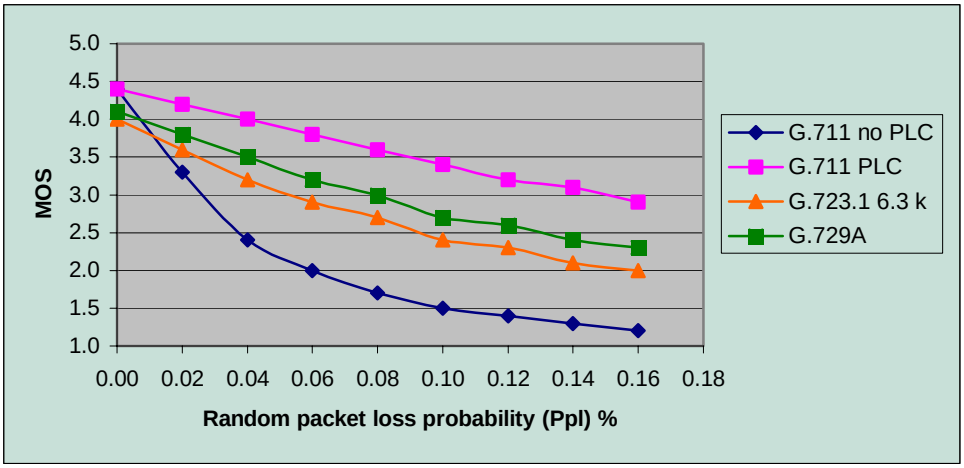

Figure 3.1. MOS under conditions of random packet loss, based on the E-Model

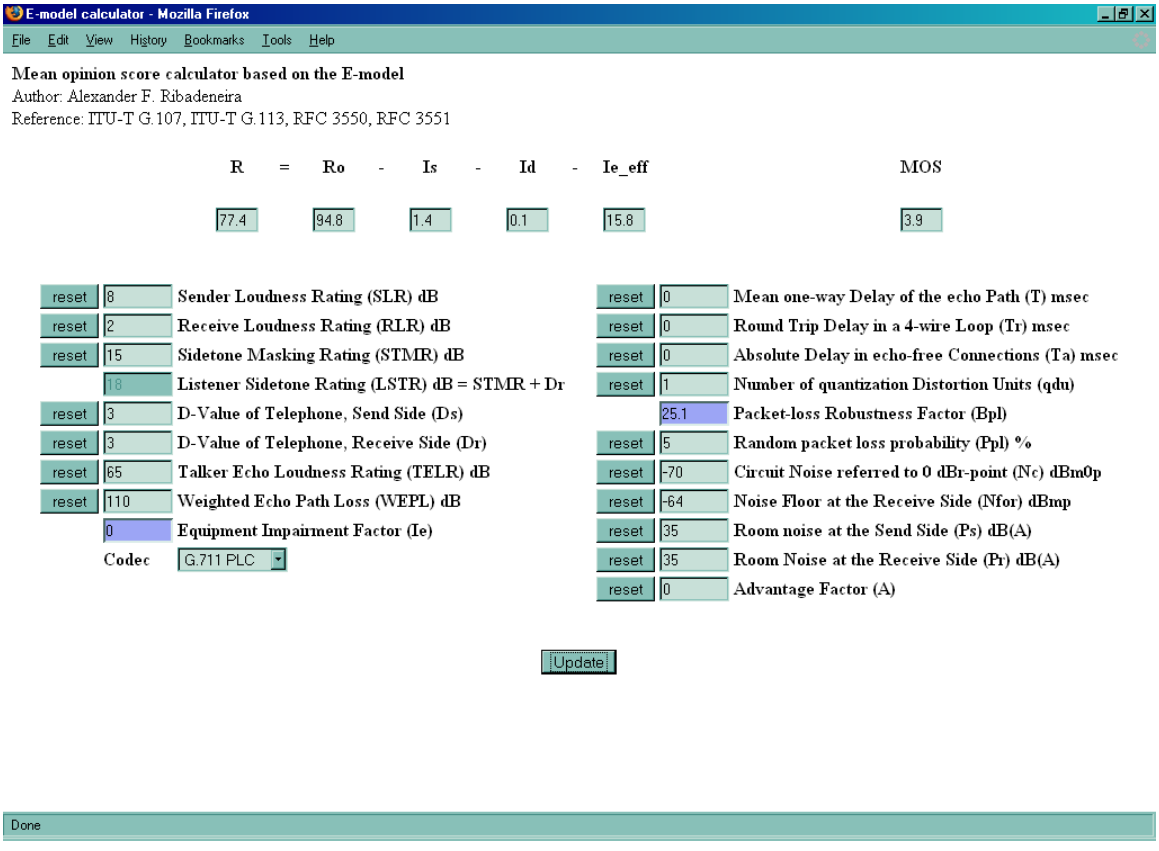

Figure 3.2. Impairment due to packet loss, JavaScript application snapshot

To calculate the MOS of a call under conditions of periodic and random loss refer to Figure 5.11 in Section 5.6. In this experiment the call was implemented with the G.711 codec operating at a transmission cycle of 20 msec. To generate the calls a script was written in Tcl using the Ixia Tcl Hal API. This script controls the Ixia Chassis to produce a RTP stream of packets. The call duration was set to one minute. The periodic and random loss was generated using an impairment generator. To measure the MOS of the call, we used SQprobe that in non-intrusive mode estimates the mean opinion score of VoIP calls traversing a network segment. SQprobe is a product of Telchemy and the test was conducted at Telchemy research labs. The code to generate the RTP stream of packets is attached to Appendix A. It can be clearly observed in Figure 5.11 that packet loss has a huge impact in the mean opinion score.

CHAPTER 4 – ANALYSIS OF INTRODUCING FEC FOR VOIP

This Chapter analyses the limitation of introducing forward error correction (FEC) in voice over IP (VoIP). We mainly evaluate two different techniques to recover from the lost of real time protocol (RTP) packets, piggybacking and Reed Solomon codes.

4.1 How to recover lost information in the event of packet loss.

If we recapitulate the things that we have learned so far, VoIP does not provide a mechanism to guarantee packet delivery because RTP and UDP do not provide error recovery mechanisms at the transport layer.

Recovery mechanisms of lost payload data have been implemented at the codec level. However, these mechanisms only recover lost information in the event of small packet loss. This is mainly because advance codecs interpolate and predict how the lost data might have look like. In addition, they implement packet loss concealment in which the receiver plays the last portion of the waveform in the event of packet loss. Packet loss concealment is a technique that considerably improves the quality of VoIP call. Error recovery at the codec level is one the most interesting areas of research.

There is mainly one technique that has been discussed over the past years in order to provide a better quality to VoIP calls, which is forward error correction (FEC) piggybacking and Reed Solomon codes. In this Chapter we examine these techniques.

4.2 Forward error correction

Forward error correction (FEC) is a mechanism that allows reliable transmissions by sending redundant data known as parity. So, the receiver is able to correct errors without retransmissions. FEC has proven to be highly efficient in scenarios where retransmissions are impossible. However, FEC increases delay in the communication process because the receiver can only start the playback process after receiving parity data. Moreover, FEC introduces the usage of additional bandwidth and in the event of congestion in a computer network it will be detrimental to the communication process. Therefore, the level of FEC that needs to be applied to a stream of RTP packet has to be addressed carefully.

There are many different FEC codes for different type of applications. Through this research we are particularly interested in systematic forward error correction codes where the original payload of RTP packets appear in the encoded output. The maximum fraction of RTP payload packets that can be recovered with a FEC scheme is determined in advance by the design of the codeword [39].

Recall that the maximum permissible delay of a VoIP call needs to be less than 150 msec for a real time communication. Our contribution is to predict the delay induced for different levels of forward correction (FEC) using piggybacking and Reed Solomon codes. We implemented a JavaScript version of the E-Model to calculate the impairment due to delay (Id) for different values of long absolute delays. Refer to appendix B for the implementation of this program. Notice that we are particularly interested in the pattern of recovery, additional bandwidth, and delay introduced when using FEC piggybacking and Reed-Solomon codes in VoIP.

4.3 Forward error correction with a piggybacking scheme

To predict the behavior of forward error correction piggybacking scheme in VoIP lets consider a RTP stream of packets at 20 msec intervals where every packets contains 20 msec of payload data. Figure 4.2 describes a piggybacking forward error correction scheme with a level of redundancy equal to two [21]. Notice that the loss of one packet does not affect the final stream because if packet n is lost in the network it can be recovered by extracting information in packet n+1. Note that with this scheme the additional bandwidth consumed is equal to 2 times the data rate of the codec plus additional headers of the TCP/IP stack. Moreover, there is an additional delay added to the communication process because if packet n is lost, it is necessary to wait for packet n+1, which of course for this particular example arrives only after 20 msec. Once packet n+1 arrives it is safe to deliver the RTP data to the nominal jitter buffer space.

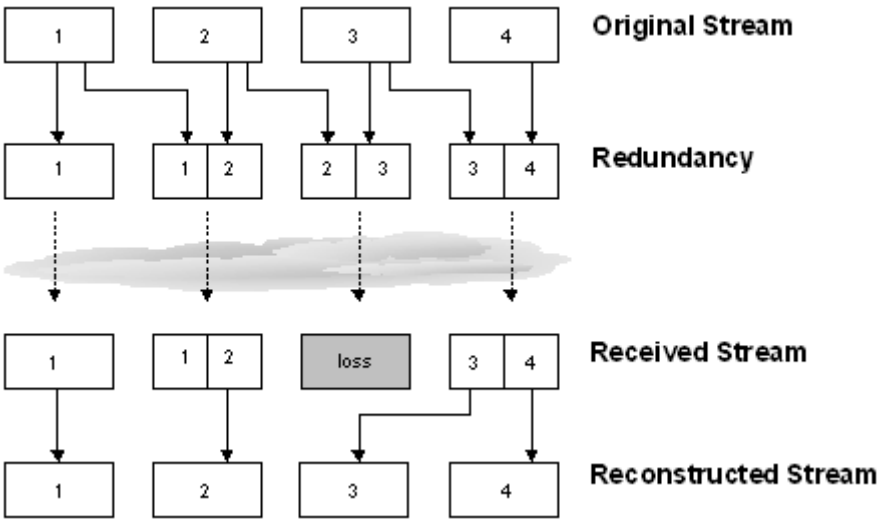

Figure 4.1. [21]. FEC piggybacking scheme with a redundancy level of two

Since the maximum permissible delay for a real time communication has to be less than 150 msec recommended by the ITU, our goal is to not exceed this maximum permissible delay. Therefore, we can generalize and state the following:

$$(\text{Piggybacking level of redundancy} - 1) * T + RTT/2 + \text{Nominal jitter buffer size} < 150 \text{ msec}$$

Where the piggybacking level of redundancy is a value equal or greater than two. However, the greater this value, the greater the bandwidth utilization. Moreover, T is the transmission cycle of the RTP stream, and RTT/2 represents the one-way delay. For this particular example we can completely recover the RTP stream if 1 of every 2 consecutive packets is lost in the network. Which is equivalent to 50% packet loss but only in patterns of 1 out of 2 consecutive packets.

If we apply FEC piggybacking scheme to a stream of RTP packets with a transmission cycle of 10 msec, the G.711 (64 kbps) codec will utilize an additional bandwidth of 80 bytes per redundant data per packet, and G.729 (8 kbps) an additional bandwidth of 10 bytes per redundant data per packet. Table 4.1 shows a bandwidth comparison on different levels of FEC piggybacking scheme for G.711 and G.729 codecs operating at 10 msec transmission cycles.

If we apply FEC piggybacking scheme to a stream of RTP packets with a transmission cycle of 20 msec, the G.711 (64 kbps) codec will utilize an additional bandwidth of 160 bytes per redundant data per packet, and G.729 (8 kbps) an additional bandwidth of 20 bytes per redundant data per packet. Table 4.2 shows a bandwidth comparison on different levels of FEC piggybacking scheme for G.711 and G.729 codecs operating at 20 msec transmission cycles.

In order to calculate the bandwidth consumption for different levels of piggybacking redundancy we need to do the following substitution to the equation shown in Section 2.10.

$$\text{Data payload per packet(bytes)} = \text{Data payload per packet(bytes)} * \text{Piggybacking level of redundancy}$$

Table 4.1. Bandwidth consumption of FEC piggybacking scheme at 10 msec intervals

Piggybacking redundancy	**Recovery pattern**	**Bandwidth consumption G.711 (kbps)**	**Bandwidth consumption G.729 (kbps)**
1 "No redundancy"	N/A	126.4	70.4
2	1 out of 2	190.4	78.4
3	2 out of 3	254.4	86.4
4	3 out of 4	318.4	94.2
5	4 out of 5	382.4	102.4
6	5 out of 6	446.4	110.4
7	6 out of 7	510.2	118.4

Table 4.2. Bandwidth consumption of FEC piggybacking scheme at 20 msec intervals

Piggybacking redundancy	**Recovery pattern**	**Bandwidth consumption G.711 (kbps)**	**Bandwidth consumption G.729 (kbps)**
1 "No redundancy"	N/A	95.2	39.2
2	1 out of 2	159.2	47.2
3	2 out of 3	223.2	55.2
4	3 out of 4	287.2	63.2
5	4 out of 5	351.2	71.2
6	5 out of 6	415.2	79.2
7	6 out of 7	479.2	87.2

It is possible to calculate the additional delay induced for certain level of piggybacking redundancy. The overall delay is defined by the formula described below.

Where T stands for transmission cycle. Table 4.3 shows the additional delay introduced for codecs operating at 10 msec and 20 msec transmission cycles respectively.

$$\textit{Piggybacking additional delay} = (\textit{Piggybacking level of redundancy} - 1) * T$$

Table 4.3 explains in detail the additional delay added to the communication process for a certain level of piggybacking redundancy. Notice that a good implementation of FEC piggybacking does not need to introduce additional delay if there is no packet loss. Therefore, the additional delay is introduced as needed. Moreover, with this scheme it is also possible to correct the payload bits from RTP packets by looking at payloads from neighbor packets at the expense of paying an additional delay. However, we do not consider this case in our research. The values presented in Table 4.3 should be interpreted as maximum additional delay under worst conditions of packet loss.

Table 4.3. Additional delay of FEC piggybacking scheme

Piggybacking redundancy	**Maximum additional delay at 10 msec intervals**	**Maximum additional delay at 20 msec intervals**
1 “No redundancy”	0	0
2	10	20
3	20	40
4	30	60
5	40	80
6	50	100
7	60	120

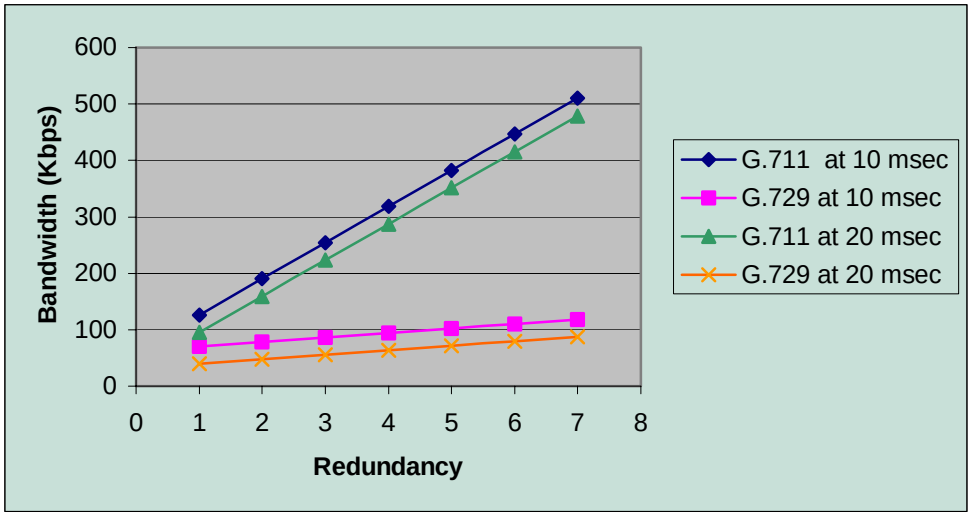

Figure 4.2. Bandwidth consumption of forward error correction piggybacking scheme

From Figure 4.2 it can be inferred that FEC piggybacking scheme has small overhead for low data rate codecs and high overhead for high data rate codecs. Therefore, in the event of network congestion it is more convenient to implement FEC piggybacking scheme for low data rate codecs only. Moreover, according to the results presented in Tables 4.1 and 4.2, it can be clearly seen that it makes more sense to implement FEC piggybacking scheme at 20 msec intervals rather than at 10 msec intervals due to the high overhead of the headers in the TCP/IP protocol stack.

Figure 4.6 represents the impairment due to delay (Id) measured with the E-Model [4]. Moreover, Figure 4.7 shows a snapshot of the JavaScript application implemented to calculate the results presented in Figure 4.6. Refer to Appendix B for an implementation of the E-Model. Notice that the overall impairment due to the delay introduced by Piggybacking scheme, plus the One-way delay, and nominal jitter buffer

delay has to be less than 150 msec to obtain high MOS values. The formula shown below describes in detail this idea.

$$(\text{Piggybacking level of redundancy} - 1) * T + RTT/2 + \text{Nominal jitter buffer size} < 150 \text{ msec}$$

Due to the high overhead in bandwidth consumption of high data codecs in a piggybacking FEC scheme, we describe its limitations in a low data rate codec only such as G.729 (8 kbps). Moreover, we also consider a transmission cycle of 20 msec, mainly because in the event of burst packet loss in a computer network the probability to lose packets at 20 msec packet intervals is less than at 10 msec intervals, and there is less bandwidth overhead at 20 msec rather than at 10 msec intervals. However, the main drawback is that the delay is two times greater. At all times we utilize the mean opinion score calculator implemented for this research.

Assume a VoIP call is placed with the G.729A codec at 20 msec intervals where the one-way delay (RTT/2) is equal to 40 msec. Also assume the receiver implements a fixed jitter buffer with a nominal value of 60 msec. So, the absolute delay in echo free connections is equal to 100 msec. Also assume that there is a consecutive loss of three packets in a stream of 50 packets per second. Therefore, the random packet loss probability is equal to 6%. Replacing these values in the mean opinion score calculator we obtain a MOS of 3.2.

Now lets examine what will be the MOS in the presence of piggybacking FEC. Since we need to recover from the lost of at least three consecutive packets in a stream, we need at least a scheme where the piggybacking level of redundancy is equal to four.

With this scheme we can recover up to three lost packets in a stream of four consecutive packets. Notice that the additional delay is equal to 60 msec. So, the absolute delay in echo free connections is equal to 160 msec. Also assume that the random packet loss probability is equal to 0%. Replacing these values in the mean opinion score calculator we obtain a MOS of 4.1.

Unfortunately, in the event of congestion the introduction of piggybacking FEC can generate additional delay induced by the computer network. Assume that this delay is equal to 30 msec. So, the absolute delay in echo free connections is equal to 190 msec. Also assume that the random packet loss probability is equal to 0%. Replacing these values in the mean opinion score calculator we obtain a MOS of 4.0.

Moreover, there is yet another issue we have to consider. Assume that the main cause of packet loss is congestion. So, the introduction of FEC could generate more packet loss. Assume that the delay in echo free connections is equal to 190 msec. Also assume that the random packet loss probability is equal to 2%. Replacing these values in the mean opinion score calculator we obtain a MOS of 3.7.

Through this theoretical analysis we conclude that the effectiveness of piggybacking FEC scheme has its limitations. It depends on the one-way delay, nominal jitter buffer size at the receiver, codec implemented, transmission cycle of the RTP stream, and congestion.

4.4 Forward Error correction with Reed Solomon codes

Reed Solomon codes are systematic block based codes that take digital data and add parity in order to recover from errors. Reed Solomon codes have been successfully

used in many systems, such as storage devices, satellite and wireless communications. The maximum number of errors that can be recovered depends on the configuration of the codeword [39].

A Reed Solomon code is specified as (n, k) with s-bit symbols. Meaning that the code takes k data symbols of s bits each, and adds parity symbols to make an n symbol codeword. Where, the number of parity symbols is defined by $n-k$, and every parity symbol contains s bits [39]. Reed Solomon codes are highly convenient for VoIP, because every RTP packet can be represented as one of the k data symbols of a codeword. In fact, Reed Solomon codes can correct up to t symbols that contain errors in a codeword, where $2t=n-k$. Figure 4.3 shows a Reed Solomon codeword.

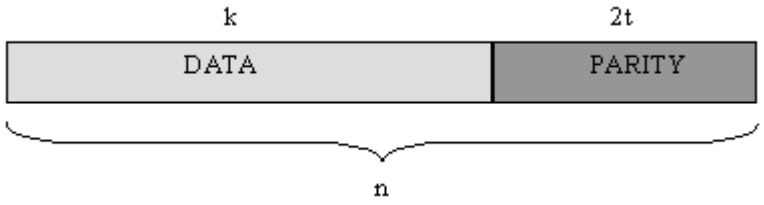

Figure 4.3. Reed Solomon codeword

Reed Solomon codes are able to correct errors at the expense of higher delay and additional bandwidth, and can only recover lost data if k out of n packets in a codeword are received at the remote end. In the analysis performed in Section 4.3 we concluded that there is high bandwidth consumption when FEC piggybacking scheme operates on high data rate codecs, the same occurs for Reed Solomon. Table 4.4 analyses the

bandwidth consumption of introducing FEC Reed Solomon codes in VoIP for different values of n and k.

Figure 4.4 describes the operation of Reed Solomon codes where n = 9 and k = 5. There are many ways to send the parity data. Figure 4.4 shows one strategy of distribution of packets and parity block data.

In Figure 4.4, T stands for transmission cycle. Notice that packet one is generated, added the appropriate headers, and then sent into the network. The same occurs for all the packets. Moreover, assume that packet 4 gets lost in the network. Then it is necessary to wait for packet 5 and the parity blocks in order to recover the lost data. With this scheme the delay generated is equal to $k \cdot T$. Notice that this strategy has a tremendous disadvantage. If packet 5 gets lost in the network, the parity data is also lost. Therefore, we need a strategy as the one presented by Schulzrinne and Jiang in [5], where the delay is equal to $n \cdot T$, this of course represents a worst condition of delay.

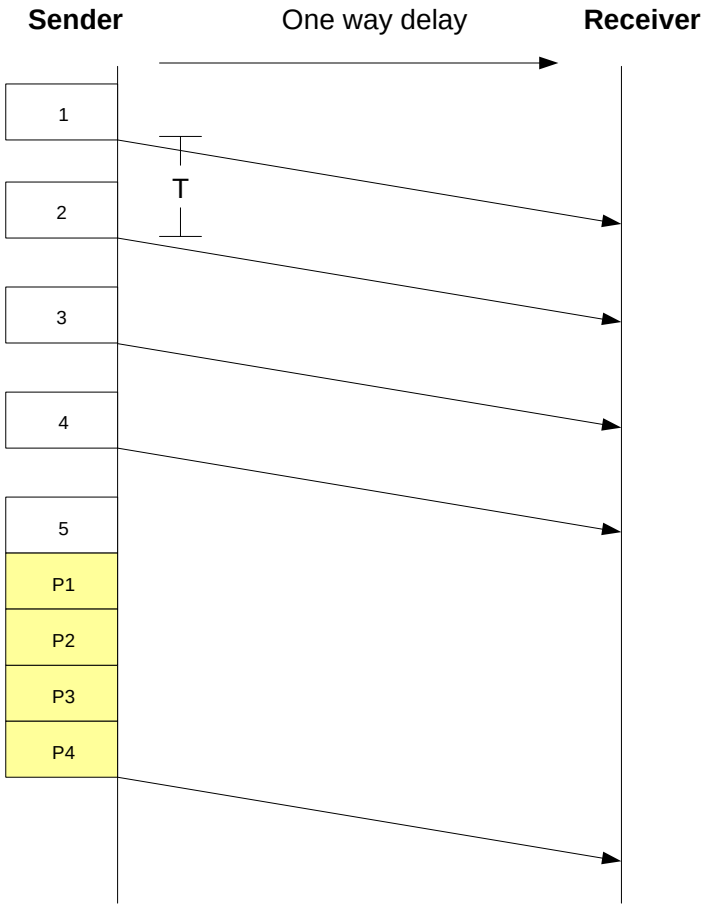

Figure 4.4. Reed Solomon codes, all parity sent in a block.

Figure 4.5, assumes a Reed Solomon strategy where the parity data is sent at the same transmission cycle as the payload data. Therefore, the additional delay is equal to $n \cdot T$. Finally, with this scheme there is an additional benefit, because subsequent payload packets from the next codeword can be sent in combination with parity blocks from the first codeword.

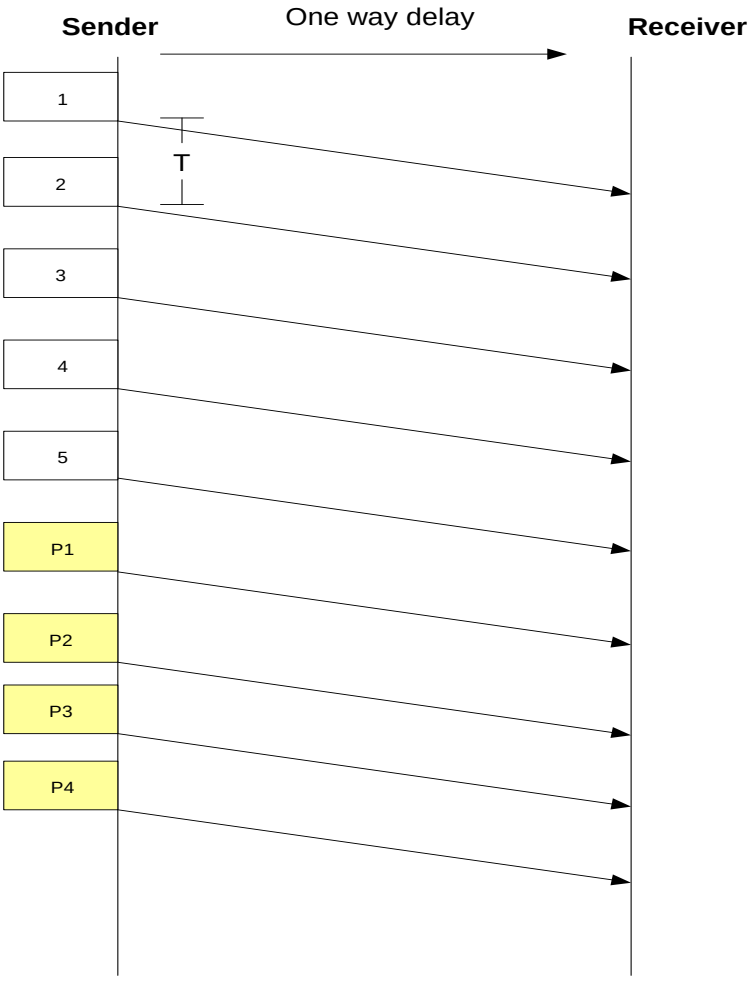

Figure 4.5. Reed Solomon codes, parity sent at T intervals.

Lets examine the bandwidth consumption and additional delay of implementing Reed Solomon codes with this strategy. The data rate and bandwidth consumption is given according to the following formulas [5].

$$\text{Rate}_{\text{Reed Solomon code}} = \text{Rate}_{\text{codec}} * \frac{n}{k}$$

$$\text{Bandwidth}_{\text{Reed Solomon code}} = \text{Rate}_{\text{Reed Solomon code}} + \frac{78\text{bytes}}{T}$$

Table 4.4. Bandwidth consumption of FEC Reed Solomon code, T=10msec

FEC Reed Solomon (n, k)	**Bandwidth consumption G.711 (kbps)**	**Bandwidth consumption G.729 (kbps)**
(4,2)	190.4	78.4
(6,2)	254.4	86.4
(5,3)	169.1	75.7
(7,3)	211.7	81.1
(9,3)	254.4	86.4
(6,4)	158.4	74.4
(8,4)	190.4	78.4
(10,4)	222.4	82.4
(12,4)	254.4	86.4
(7,5)	152.0	73.6
(9,5)	177.6	76.8
(11,5)	203.2	80.0
(13,5)	228.8	83.2
(15,5)	254.4	86.4

Tables 4.4 and 4.5 represent the additional bandwidth of introducing FEC Reed Solomon codes in VoIP with a Transmission cycle of 10 and 20 msec respectively. Note that we do not analyze the behavior at 30 msec or greater because the induced delay is too high. Remember that a good implementation of Reed Solomon should not introduce a delay greater than 150 msec.

From Tables 4.4 and 4.5 it can be inferred that FEC Reed Solomon codes has small overhead for low data rate codecs and high overhead for high data rate codecs. Therefore, in the event of network congestion it is more convenient to implement FEC Reed Solomon codes for low data rate codecs only. Moreover, it can be clearly seen that at 10 msec packet intervals the delay introduced by the FEC Reed Solomon codes is half than at 20 msec intervals. However, there is yet another problem, the bandwidth is almost

two times greater due to the high overhead of the headers in the TCP/IP protocol stack. Consequently, in the event of congestion it will be detrimental to the communication.

Table 4.5. Bandwidth consumption of FEC Reed Solomon code, T=20msec

FEC Reed Solomon (n, k)	**Bandwidth consumption G.711 (kbps)**	**Bandwidth consumption G.729 (kbps)**
(4,2)	159.2	47.2
(6,2)	223.2	55.2
(5,3)	137.9	44.5
(7,3)	180.5	49.9
(9,3)	223.2	55.2
(6,4)	127.2	43.2
(7,5)	120.8	42.4

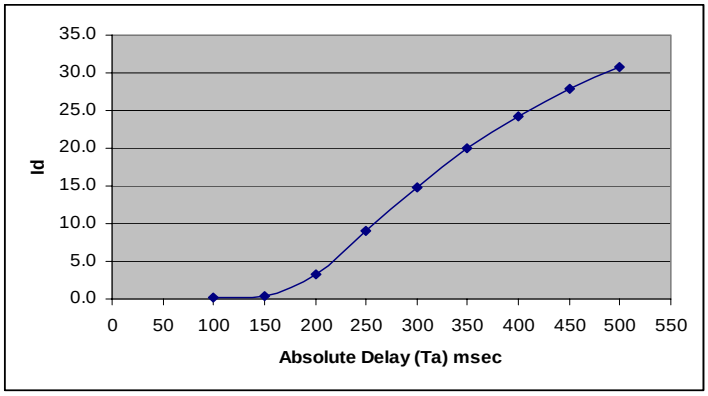

Figure 4.6. Impairment due to delay, based on the E-Model.

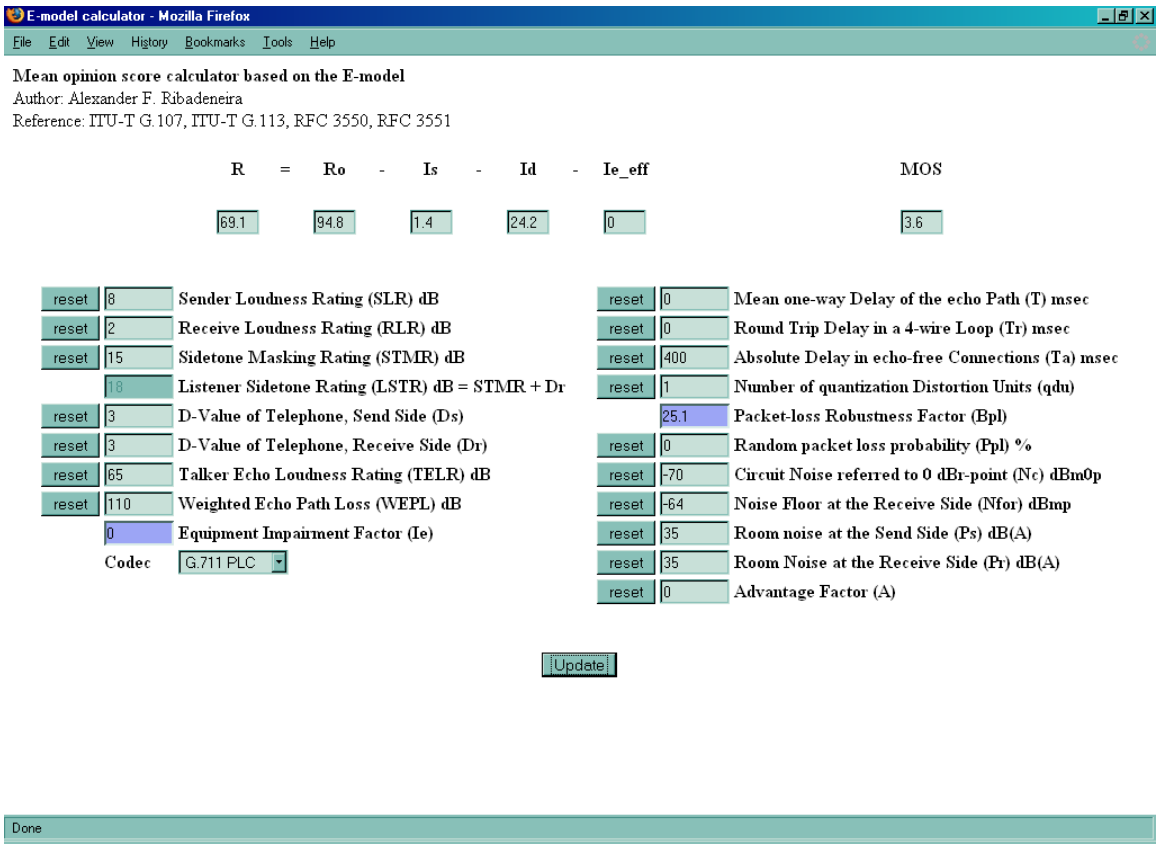

Figure 4.7. Impairment due to delay, JavaScript application snapshot

Figure 4.6 represents the impairment due to delay (Id) measured with the E-Model [4]. Moreover, Figure 4.7 shows a snapshot of the JavaScript application implemented to calculate the results presented in Figure 4.6. Refer to Appendix B for a detail explanation of the E-Model. Notice that the overall impairment due to the delay introduced by Reed Solomon codes, plus the One-way delay, and nominal jitter buffer delay has to be less than 150 msec to obtain high MOS values. The formula shown below describes in detail this idea.

$$n * T + RTT/2 + \text{Nominal jitter buffer size} < 150 \text{ msec}$$

Due to the high overhead in bandwidth consumption of high data codecs in a Reed Solomon FEC scheme, we describe its limitations in a low data rate codec only such as G.729 (8 kbps). Moreover, we also consider a transmission cycle of 20 msec, mainly because in the event of burst packet loss in a computer network the probability to lose packets at 20 msec packet intervals is less than at 10 msec intervals, and there is less bandwidth overhead at 20 msec rather than at 10 msec intervals. However, the main drawback is that the delay is two times greater. At all times we utilize the mean opinion score calculator implemented for this research.

Assume a VoIP call is placed with the G.729A codec at 20 msec intervals where the one-way delay (RTT/2) is equal to 40 msec. Also assume the receiver implements a fixed jitter buffer with a nominal value of 60 msec. So, the absolute delay in echo free connections is equal to 100 msec. Also assume that there is a consecutive loss of three packets in a stream of 50 packets per second. Therefore, the random packet loss probability is equal to 6%. Replacing these values in the mean opinion score calculator we obtain a MOS of 3.2.

Now lets examine what will be the MOS in the presence of Reed Solomon FEC. Since we need to recover from the loss of at least 3 consecutive packets in a stream, we need at least a scheme where n = 9 and k =3. With this scheme we can recover up to three lost blocks in a codeword. Notice that the additional delay $n \cdot T$ is equal to 180 msec. So, the absolute delay in echo free connections is equal to 280 msec. Also assume that the random packet loss probability is equal to 0%. Replacing these values in the mean opinion score calculator we obtain a MOS of 3.6.

Unfortunately, in the event of congestion the introduction of Reed Solomon FEC can generate additional delay induced by the computer network. Assume that this delay is equal to 30 msec. So, the absolute delay in echo free connections is equal to 300 msec. Also assume that the random packet loss probability is equal to 0%. Replacing these values in the mean opinion score calculator we obtain a MOS of 3.4.

Moreover, there is yet another issue we have to consider. Assume that main cause of packet loss is congestion. So, the introduction of FEC could generate more packet loss. Assume that the delay in echo free connections is equal to 300 msec. Also assume that the random packet loss probability is equal to 2%. Replacing these values in the mean opinion score calculator we obtain a MOS of 3.0.

Through this theoretical analysis we conclude that the effectiveness of Reed Solomon FEC scheme has its limitations. It depends on the one-way delay, nominal jitter buffer size at the receiver, codec implemented, transmission cycle of the RTP stream, and congestion.

CHAPTER 5 – EXPERIMENTS AND RESULTS

This chapter describes the experiments set up during this research. These experiments are necessary to have a better understanding on how the quality of VoIP calls are affected under the event of distinct forms of impairments, such as: constant delay, random delay, Gaussian delay with packet reordering, Gaussian delay without packet reordering, and packet loss. In addition, we also measure the mean opinion score (MOS) of calls when phones use a fixed and an adaptive jitter buffer respectively.

5.1 Degradation due to loss of consecutive audio.

To understand the impact that loss of consecutive audio has in the degradation of listening quality, it was necessary to use an audio editor program, “Cool edit Pro”. We edit an audio reference file of 5 seconds long, and generate silence of 5, 10, 20 and 40 msec respectively. Moreover, using ear human perception, we will define the threshold in milliseconds at which click and pops are generated. The audio reference file is in PCM format at 8 kHz, 16 bit, mono.

Figure 5.1 shows the original audio and waveforms impaired at 2.6 seconds. The main result of this experiment is that even for an impairment silence of 5 msec, click and pops are generated. Therefore, the loss of 5 msec of audio is an annoying event that affects the quality of a VoIP call.

Figure 5.1. Call quality degradation

5.2 Call degradation on Cisco ATA 188 in the event of distinct forms of delay

To understand the impact that network impairments, such as distinct forms of delay has in the quality of VoIP calls. It was necessary to setup the network environment described in Figure 5.2. VoIP phones with Session Initiation Protocol (SIP) firmware will register to a SIP proxy Server (Ondo SIP server). The Cisco ATAs 188 run a software version 3.02.01 (050616A). An impairment emulator “Spirent IP Wave fx” will generate fixed delay, uniform delay, Gaussian delay, and jitter to a stream of RTP packets.

User A “the caller” will play a previously recorded audio file using the English language. Finally, user B “the callee” at the remote end using ear human perception determines the threshold of impairments at which the quality of the call degrades because click and pops are introduced in the channel.

Figure 5.2 describes the network topology for the experiment. User A calls user B phone number. Since both phones are registered to the SIP proxy server (Ondo SIP server), a SIP setup call is established. Moreover, codec negotiation is forced to be G.711 PCMA with silence suppression turned-off at 20 msec transmission cycle. Once user B answers, user A starts playing a prerecorded audio file. The RTP packets traveling across the network from A to B will be delayed with the impairment generator.

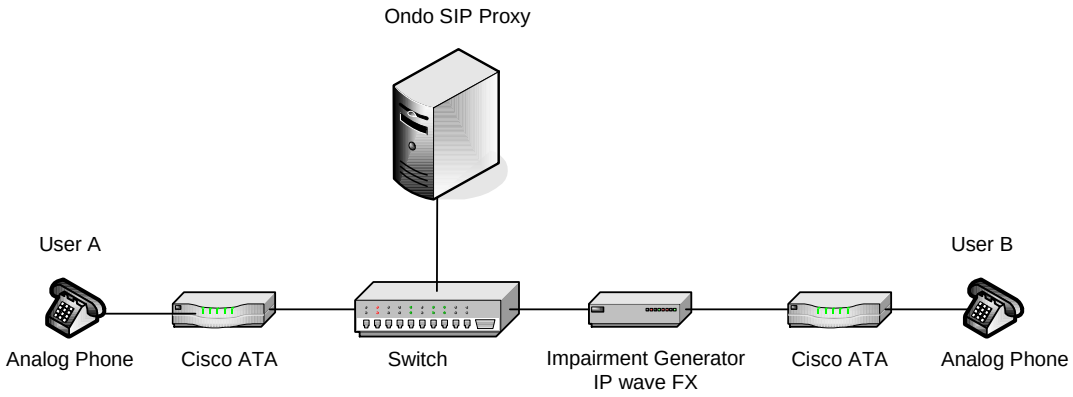

Figure 5.2. SIP setup for Cisco ATA 188 phones with an impairment generator

The results obtained are shown below.

Fixed delay - Listening quality is good for every value of fixed delay, even for 10.000 msec delay. However, this affects conversational quality.

Uniform delay - The impairment generator randomly applies delays between minimum and maximum values. Uniform delay causes packets to be out of sequence due to variable delays being applied. If the minimum value of uniform delay is set to 0 msec then the

maximum value of uniform delay at which clicks and pops are noticed is around 280 msec.

Gaussian delay - The impairment generator applies delays that resemble a Gaussian distribution between a minimum and maximum values, which represent the -3 and +3 standard deviation respectively. Due to the randomly applied Gaussian delay, packets may become out of order. If the minimum value of Gaussian delay is set to 0 msec then the maximum value of Gaussian delay at which clicks and pops are noticed is around 320 msec.

Jitter – The impairment generator applies delays that resemble a Gaussian distribution between a minimum and maximum values, which represent the -3 and +3 standard deviation respectively. But, in this case packets are forced to maintain order. If the minimum value of Gaussian delay is set to 0 msec then the maximum value of Gaussian delay at which clicks and pops are noticed is around 550 msec. Notice that the playback interval of packets from the impairment generator with a jitter function is most of the times 0 msec. Therefore, this resembles a burst of packets event which also generates jitter buffer discards. Discards are generated when there is excessive inter packet delay or the burst size is greater than the maximum jitter buffer size. Figure 5.3 shows a 10 seconds snapshot of the jitter induced by the impairment generator for this particular experiment. It was obtained from a one-minute call using G.711 PCMA codec at 20 msec intervals with voice activity detection turned-off. The packet intervals of the output stream were captured with ethereal network protocol analyzer. Finally this event can occur when routers hold packets in its buffers due to queuing delay.

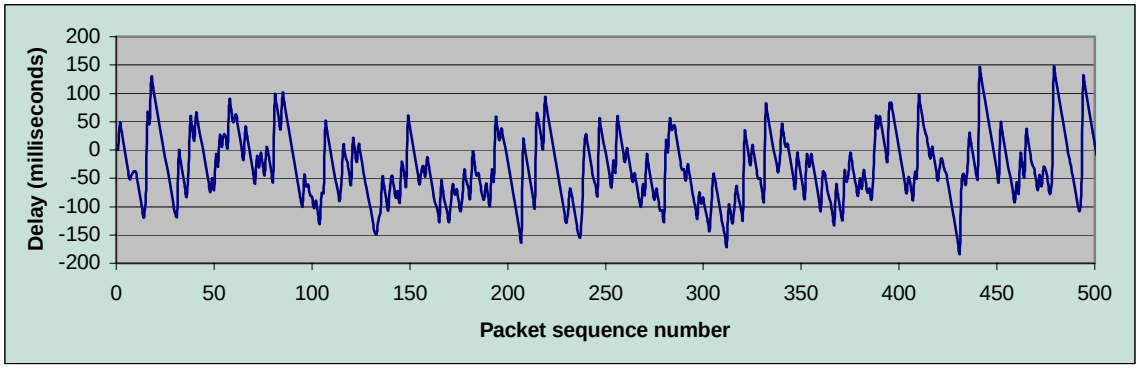

Figure 5.3. Jitter without re-ordering for a Gaussian delay between 0 and 550 msec.

5.3 Call degradation on Grand Stream ATA in the event of distinct forms of delay

To understand the impact that network impairments, such as distinct forms of delay has in the quality of VoIP calls. It was necessary to setup the network environment described in Figure 5.4. VoIP phones with Session Initiation Protocol (SIP) firmware will register to a SIP proxy Server (Ondo SIP server). The Grandstream phones HT286 run a software version 1.0.5.0. An impairment emulator "Spirent IP Wave fx" will generate fixed delay, uniform delay, Gaussian delay, and jitter to a stream of RTP packets.

User A "the caller" will play a previously recorded audio file using the English language. Finally, user B "the callee" at the remote end using ear human perception determines the threshold of impairments at which the quality of the call degrades because click and pops introduced in the channel.

Figure 5.4 describes the network topology for the experiment. User A calls user B phone number. Since both phones are registered to the SIP proxy server (Ondo SIP server), a SIP setup call is established. Moreover, codec negotiation is forced to be G.711 PCMA without silence suppression at 20 msec transmission cycle. Once user B answers, user A starts playing a prerecorded audio file. RTP packets will be sent in both directions

because voice activity detection is turned-off. The RTP packets traveling across the network from A to B will be delayed with the impairment generator.

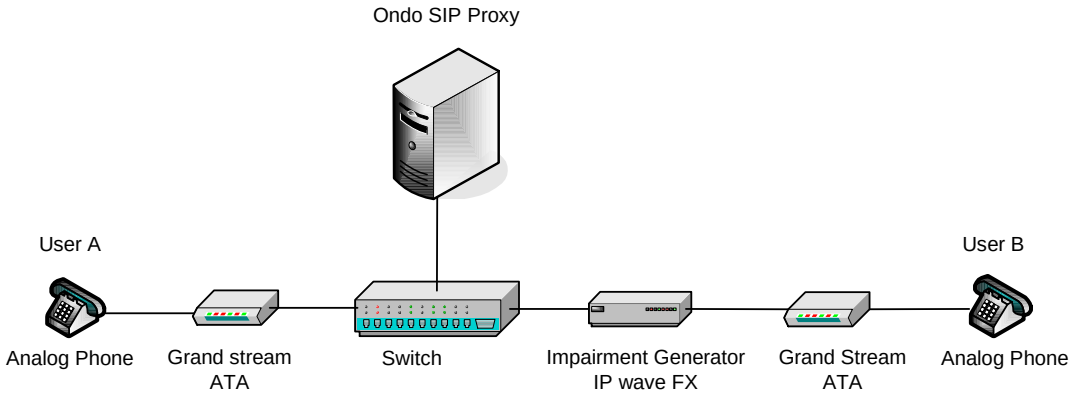

Figure 5.4. SIP setup for Grand stream phones with an impairment generator

The results obtained are shown below.

Fixed delay - Listening quality is good for every value of fixed delay, even for 10.000 msec delay. However, this affects conversational quality.

Uniform delay – The impairment generator randomly applies delays between minimum and maximum values. Uniform delay causes packets to be out of sequence due to variable delays being applied. If the minimum value of uniform delay is set to 0 msec then the maximum value of uniform delay at which clicks and pops are noticed is around 50 msec.

Gaussian delay – The impairment generator applies delays that resemble a Gaussian distribution between a minimum and maximum values, which represent the -3 and +3 standard deviation. Due to the randomly applied Gaussian delay, packets may become out of order. If the minimum value of Gaussian delay is set to 0 msec then the maximum value of Gaussian delay at which clicks and pops are noticed is around 80 msec.

Jitter – The impairment generator applies delays that resemble a Gaussian distribution between a minimum and maximum values, which represent the -3 and +3 standard deviation respectively. But, in this case packets are forced to maintain order. If the minimum value of Gaussian delay is set to 0 msec then the maximum value of Gaussian delay at which clicks and pops are noticed is around 100 msec. Notice that the playback interval of packets from the impairment generator with a jitter function is most of the times 0 msec. Therefore, this resembles a burst of packets event which also generates jitter buffer discards. Discards are generated when there is excessive inter packet delay or the burst size is greater than the maximum jitter buffer size. Figure 5.3 shows a 10 seconds snapshot of the jitter induced by the impairment generator for this particular experiment. It was obtained from a one-minute call using G.711 PCMA codec at 20 msec intervals with voice activity detection turned-off. The packet intervals of the output stream were captured with ethereal network protocol analyzer. Finally this event can occur when routers hold packets in its buffers due to queuing delay.

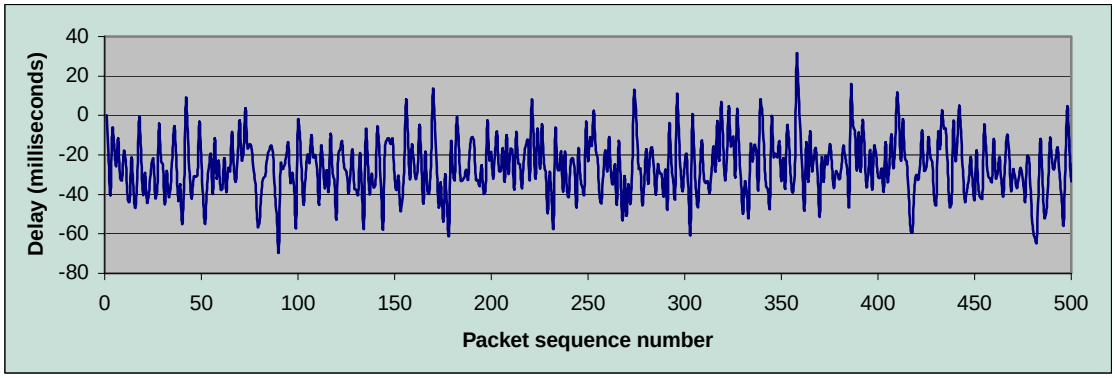

Figure 5.5. Jitter without re-ordering for a Gaussian delay between 0 and 100 msec.

5.4 Call quality degradation with a fixed jitter buffer.

To understand the impact that a fixed jitter buffer has in the quality of a VoIP call in the presence of impairments such as Gaussian delay, it was necessary to setup the network environment described in Figure 5.6. Generic VoIP analog telephone adapters were forced to operate with a fixed jitter buffer, with the following configuration, 40 msec of minimum jitter buffer size, 40 msec of nominal jitter buffer size, and 80 msec of maximum jitter buffer size. An impairment emulator “The cloud” will generate Gaussian delay with a mean of 100 msec and standard deviations that range from 5 to 100 msec. The codec for the call was set to G.711a at 20 msec packet intervals without voice activity detection. A computer will play a previously recorded audio file in the English language. The format of the audio file is PCM at 8 kHz, 16 bits, mono. The length of the reference audio file is 30 seconds. The reference audio file is played through the computer sound card speaker port. Consequently, the impaired audio is received using a computer sound card recording port. Moreover, Ethereal captures the timing of packets traversing the network. Finally, the received audio file is compared against the original audio file using perceptual evaluation of speech quality (PESQ) algorithm P.862 [6].

Figure 5.7 shows the original reference audio file and impaired audio files recorded with the audio editor program, where D stands for delay in milliseconds, and SD stands for standard deviation in milliseconds. Moreover, SD refers to the first standard deviation of a Gaussian normal distribution. Meaning that 68% of random delay values generated by the impairment generator fall between –1 and +1 standard deviation from the mean of 100 msec. 95% fall between –2 and +2 standard deviations, and 99% fall between –3 and +3 standard deviations.

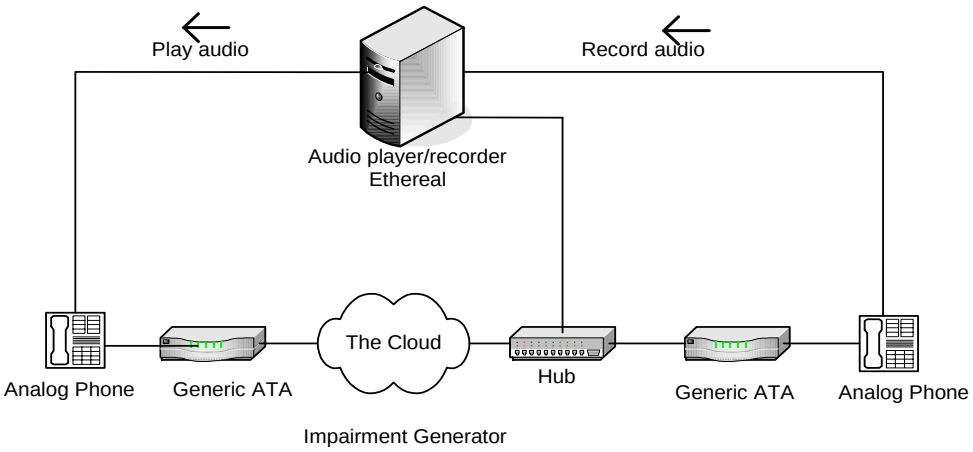

Figure 5.6. Setup to measure call quality degradation in the event of Gaussian delay.

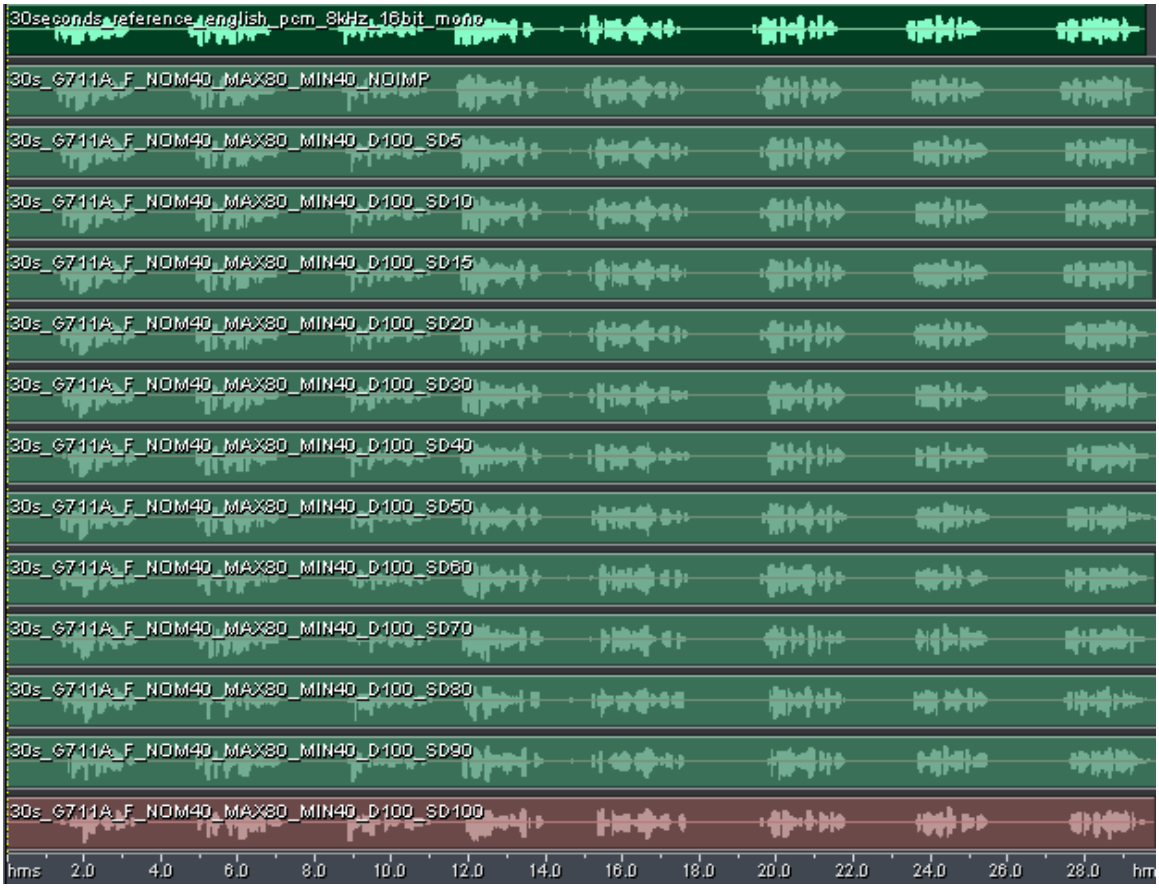

Figure 5.7. Fixed jitter buffer impaired waveforms.

It can be clearly seen in Figure 5.7 that for large values of standard deviations there is high degradation of the waveform. Consequently, PESQ will compute lower mean opinion scores because the output waveform does not look like as similar as the original waveform. Moreover, during some periods of time the output waveform suffers from the loss of consecutive audio. Recall that according to the experiments performed in Section 5.1, the loss of 5 msec of consecutive audio is an annoying event. The only technique that could help prevent from this impairment is to increase the size of the maximum and nominal jitter buffers. However, the greater the size of these buffers, the greater the fixed delay, and that itself generates another impairment known as absolute delay in echo-free connections (Ta). The jitter buffer discards were measured with Telchemy's VQmon. Note that when there are no impairments none of the 1500 RTP packets are discarded by the jitter buffer. The results obtained during this experiment are shown in Table 5.1 and Figure 5.8.

Table 5.1. Mean opinion score and discards for a fixed jitter buffer

Impairment	**MOS (ITU P.862)**	**Discards**
No impairment	4.368	0
Delay 100 msec, standard deviation 5 msec	4.363	1
Delay 100 msec, standard deviation 10 msec	4.363	3
Delay 100 msec, standard deviation 15 msec	3.607	49
Delay 100 msec, Standard deviation 20 msec	2.961	114
Delay 100 msec, Standard deviation 30 msec	2.446	285
Delay 100 msec, Standard deviation 40 msec	1.928	434
Delay 100 msec, Standard deviation 50 msec	1.644	518
Delay 100 msec, Standard deviation 60 msec	1.501	626
Delay 100 msec, Standard deviation 70 msec	1.384	652
Delay 100 msec, Standard deviation 80 msec	1.287	704
Delay 100 msec, Standard deviation 90 msec	1.225	724
Delay 100 msec, Standard deviation 100 msec	1.212	724

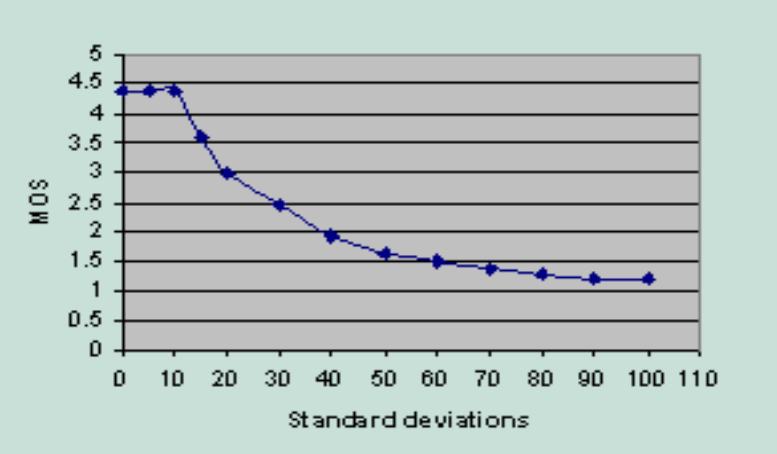

Figure 5.8. Mean Opinion Score with a fixed jitter buffer.

5.5 Call quality degradation with an adaptive jitter buffer.

To understand the impact that an adaptive jitter buffer has in the quality of a VoIP call in the presence of impairments such as Gaussian delay, it was necessary to setup the network environment described in Figure 5.6. Generic VoIP analog telephone adapters were forced to operate with an adaptive jitter buffer, with the following configuration, 20 msec of minimum jitter buffer size, 60 msec of nominal jitter buffer size, and 120 msec of maximum jitter buffer size. An impairment emulator “The cloud” will generate Gaussian delay with a mean of 100 msec and standard deviations that range from 5 to 100 msec. The codec for the call was set to G.711A at 20 msec packet intervals without voice activity detection. A computer will play a previously recorded audio file in the English language. The format of the audio file is PCM at 8 kHz, 16 bits, mono. The length of the reference audio file is 30 seconds. The reference audio file is played through the computer sound card speaker port. Consequently, the impaired audio is received using a computer sound card recording port. Moreover, Ethereal captures the timing of packets

traversing the network. Finally, the received audio file is compared against the original audio file using perceptual evaluation of speech quality (PESQ) algorithm P.862 [6].

Figure 5.9 shows the original reference audio file and impaired audio files recorded with the audio editor program, where D stands for delay in milliseconds, and SD stands for standard deviation in milliseconds. Moreover, SD refers to the first standard deviation of a Gaussian normal distribution. Meaning that 68% of random delay values generated by the impairment generator fall between –1 and +1 standard deviation from the mean of 100 msec. 95% fall between –2 and +2 standard deviations, and 99% fall between –3 and +3 standard deviations.

Figure 5.9. Adaptive jitter buffer impaired waveforms.

It can be clearly seen in Figure 5.9 that for large values of standard deviations there is high degradation of the waveform. Consequently, PESQ will compute lower mean opinion scores because the output waveform does not look like as similar as the original waveform. Moreover, during some periods of time the output waveform suffers from the loss of consecutive audio. Recall that according to the experiments performed in Section 5.1, the loss of 5 msec of consecutive audio is an annoying event. The only technique that could help prevent from this impairment is to increase the size of the maximum and nominal jitter buffers. However, the greater the size of these buffers, the greater the fixed delay, and that itself generates another impairment known as absolute delay in echo-free connections (Ta). The jitter buffer discards were measured with Telchemy's VQmon. Note that when there are no impairments none of the 1500 RTP packets are discarded by the jitter buffer. The results obtained during this experiment are shown in Table 5.2 and Figure 5.10.

Table 5.2. Mean opinion score and discards for an adaptive jitter buffer

Impairment	**MOS (ITU P.862)**	**Discards**
No impairment	4.369	0
Delay 100 msec, Standard deviation 5 msec	4.370	1
Delay 100 msec, Standard deviation 10 msec	4.362	5
Delay 100 msec, Standard deviation 15 msec	4.363	13
Delay 100 msec, Standard deviation 20 msec	4.297	15
Delay 100 msec, Standard deviation 30 msec	3.455	78
Delay 100 msec, Standard deviation 40 msec	2.872	258
Delay 100 msec, Standard deviation 50 msec	2.738	347
Delay 100 msec, Standard deviation 60 msec	2.233	425
Delay 100 msec, Standard deviation 70 msec	2.065	530
Delay 100 msec, Standard deviation 80 msec	1.861	573
Delay 100 msec, Standard deviation 90 msec	1.876	657
Delay 100 msec, Standard deviation 100 msec	1.683	722

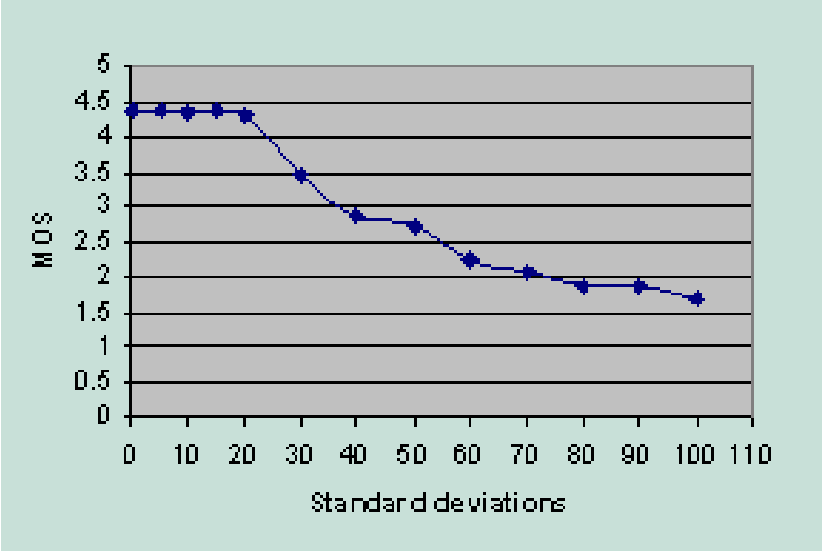

Figure 5.10. Mean Opinion Score for an adaptive jitter buffer.

5.6 MOS under the event of periodic and random lost (non-intrusive technique).

To understand the impact that periodic and random loss has in the quality of a VoIP call, a script that simulates a RTP stream of packet flowing in two directions was created using the Ixia Tcl Hal API. This script controls the Ixia Chassis to produce a RTP stream of packets with G.711 codec operating at a transmission cycle of 20 msec.

The call duration was set to one minute. The periodic and random loss was generated using an impairment generator “IP Wave Fx”. The mean opinion score was measured with SQprobe that works in non-intrusive mode. SQprobe is a product of Telchemy and the test was conducted at Telchemy research labs. The code to generate the RTP stream of packets is attached in Appendix A. Table 5.3 and Figure 5.11 present the results obtained during this experiment.

Table 5.3. MOS of G.711 under conditions of periodic and random loss

% Packet Lost	MOS-LQ Periodic Lost	MOS-LQ Random Lost
0	4.2	4.2
1	4.1	4.1
2	4.1	4
3	4.1	3.8
4	4	3.9
5	4	3.8
10	3.7	3.3
15	3.4	3.1
20	3.2	2.8
25	2.9	2.6
30	2.7	2.3
35	2.5	1.8

It can be clearly seen in Figure 5.11 that packet loss has a huge impact in the mean opinion score MOS of a VoIP call. Moreover, random loss is more detrimental to VoIP than periodic loss.

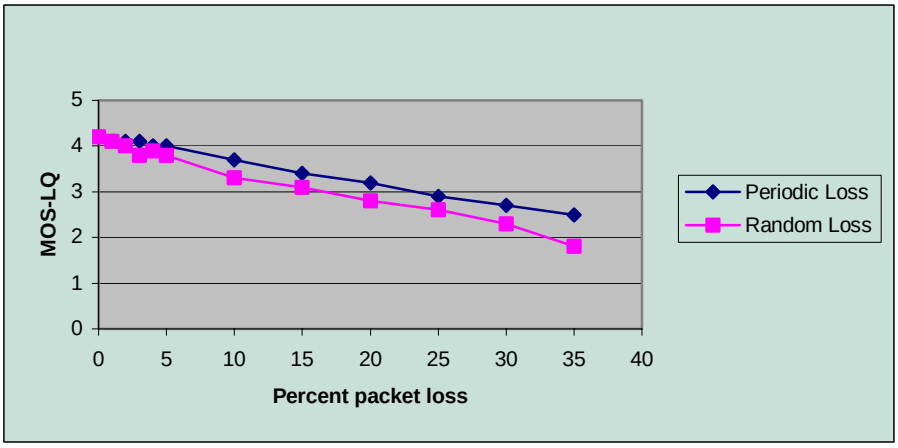

Figure 5.11. MOS of G.711 under conditions of periodic and random loss

CHAPTER 6 – CONCLUSION

Providing a better quality to VoIP calls under conditions of delay, jitter and packet loss still remains a major problem that needs to be addressed for next generation of VoIP services.

The quality of a call is not affected by the signaling protocol. However, it has a direct impact in the call setup time. In our experiments we found that call setup time in a network without impairments is approximately 250 msec using the SIP protocol.

Transmitting at small packet intervals leads to a huge waste of bandwidth resources. In fact, this is due to the overhead of the headers in the TCP/IP stack, which is 78 bytes per packet on Ethernet networks. In fact, for small data rate codecs every packet contains more headers than actual payload data. This overhead can be as high as 79.59% for G.729 (8 kbps) codec transmitting at 20 msec intervals and 88.63% for the same codec transmitting at 10 msec intervals. Moreover, There is more probability to lose packets at 10 msec intervals rather than at 20 msec intervals. Therefore, in the event of transient network problems it is more convenient sent data at 20 msec intervals. In addition RTP streams consume more bandwidth at 10 msec intervals rather than at 20 msec intervals.

The quality of a call is degraded just for the fact of using low bit rate codecs. In fact, for perfect network conditions, meaning that there is no packet loss or delay, G.711 and G.729A provide a MOS of 4.4 and 4.1 respectively. Furthermore, in the event of network congestion it makes more sense to implement a low bit rate codec such as

G.729A (8 kbps) rather than a high bit rate codec. In addition, the voice activity detection feature should always be used, because it reduces congestion.

The jitter buffer can only compensate for the event of variable inter-packet delay. Moreover, we confirm the fact that adaptive jitter buffers have better performance than fixed jitter buffers. If the nominal jitter buffer size is too high this introduces unnecessary latency which is a considerably impairment. Moreover, the maximum jitter buffer size also plays and important role in the quality of a call because if packets arrive too late the effect will be discard by the jitter buffer, and if the burst of packets that arrives at the receiver is greater than the maximum jitter buffer size, the effect is again packet discards. Recall that packet discard is equivalent to packet loss.

Fixed delay is not detrimental to the listening quality of a call. However, for values greater than 150 msec it has a direct impact in the conversational quality. From all forms of delay that we generated in our experiments, random delay is the most detrimental to VoIP. Gaussian delay without reordering packets is less detrimental to a VoIP call than Gaussian delay with packet reordering. Moreover, the quality of a VoIP call is different from vendor to vendor, this is mainly because vendors use different hardware, software, and jitter buffer algorithms.

Even though there has been amazing progress in the field of codec design such as G.711 PLC and G.729A that provide a MOS of 3.9 and 3.3 respectively when the random packet loss probability is equal to 5%. Error recovery at the codec level has only an acceptable performance up to a certain extent of packet loss.

With forward error correction the receiver is able to recover from packet loss without retransmission. Therefore, VoIP calls can have a better mean opinion score.

However, FEC increases delay in the communication, and according to the mean opinion score calculator based on the E-Model implemented on this research, this is a significant impairment that affects the conversational quality of a call. Moreover, the effectiveness of FEC depends on the one-way delay, nominal jitter buffer size at the receiver, codec implemented, transmission cycle of the RTP stream, and congestion in a computer network.

The introduction of forward error correction has two direct negative impacts. It introduces an additional delay and consumes bandwidth resources. In fact, the greater the level of redundancy, the greater the delay and bandwidth utilization, and in the event of network congestion it will be detrimental to the communication.

FEC should only be implemented in scenarios where retransmissions are impossible. A good implementation of FEC piggybacking does not need to introduce delay if there is no packet loss. Therefore, the additional delay should be introduced as needed. FEC piggybacking scheme has small overhead for low data rate codecs only and should only be implemented in high data rate codecs in the event of transient network problems. Moreover, FEC piggybacking introduces less overhead at 20 msec intervals rather than at 10 msec due to the high overhead of the headers in the TCP/IP protocol stack

Reed Solomon codes should only be implemented on low data rate codecs. At 10 msec intervals the delay introduced by the FEC Reed Solomon codes is half the time than the delay generated at 20 msec intervals, yet there is another problem because for the G.729 (8 kbps) codec the bandwidth utilization implementing a Reed Solomon code is two times greater due to the high overheads of the TCP/IP stack.

To recover from the lost of consecutive RTP packets at 20 msec intervals, FEC Reed Solomon codes introduce three times more delay than FEC piggybacking scheme. Therefore, due to the nature of the application FEC piggybacking outperforms Reed Solomon.

The delay induced by a forward error correction technique plus the one-way delay and nominal jitter buffer delay should be smaller than 150 msec for a real time communication. However, we found in our experiments that delays greater than 150 msec are acceptable if the delay is constant. Constant delay does not affect the listening quality of a call; it only affects the conversational quality. This is known as a one-way communication.

The output of the E-Model has not been completely verified because of the large number of possible combinations of input parameters that affect the overall quality of a call. In fact, modifications to the E-Model are currently under study. Moreover, the values for the input parameters that we use to develop the mean opinion score calculator application can be subject to change in the near future. Therefore, readers should use discretion at interpreting the results presented through this research.

In the future protocol such as RTCP-XR that provide reception quality feedbacks of RTP will play an important role to enhance the quality of VoIP calls because appropriate reports could provide real time feedback of the best method used for recovery, such as: transmit data at lower rates, change to a high or low bit rate codec with FEC, RUDP, or other technique if it provides better performance. Note that, protocols at the transport layer that request retransmission of lost packet are highly inefficient in

scenarios with high RTT, but the performance of these protocols for small values of RTT is still possible.

BIBLIOGRAPHY

1. H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson. RFC 3550. RTP: A Transport Protocol for Real-Time Applications. July 2003.
2. H. Schulzrinne, S. Casner. RFC 3551. RTP Profile for Audio and Video Conferences with minimal control. July 2003.
3. H. OOUCHI, T. TAKENAGA, H. SUGAWARA and M. MASUGI. Study on Appropriate Voice Data Length of IP Packets for VoIP Network adjustment. Global Telecommunications Conference, 2002. Volume 2, on page(s): 1618-1622.
4. ITU-T Recommendation G.107. The E-model, a computational model for use in transmission planning. March 2003.
5. W. Jiang and H. Schulzrinne. Comparison of FEC and codec robustness on VoIP quality and bandwidth efficiency. Submitted to World Scientific. June 5, 2002.
6. ITU-T P.862. Perceptual evaluation of speech quality (PESQ): An objective method for end-to-end speech quality assessment of narrow-band telephone networks and speech codecs. Feb 1, 2001.
7. G. Bai, K. Long, W. Wang, S. Cheng. A new reliable signaling transport protocol RSTP for voice over IP. Fifth Asia-Pacific Conference. 1999. Issue 1999, Volume 1, on page(s): 263 – 266.
8. ITU-T G.113. Transmission impairments due to speech processing. February 2001.
9. T. Friedman, R. Caceres, A. Clark. RFC 3611. RTP Control Protocol Extended Reports (RTCP XR). November 2003.
10. J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler. RFC 3261. SIP: Session Initiation Protocol. June 2002.
11. S. Cherry. Seven myths about voice over IP. Spectrum, IEEE. Volume 42, Issue 3, on page(s): 52 – 57. March 2005.

12. P. Pragtong, K. Ahmed, and T. Erke. Analysis of speech data rate in voice over IP conversation. TENCON 2004. 2004 IEEE Region 10 Conference. 2004. Volume A, on page(s): 143 – 146.
13. H. Chong, H. Matthews. Comparative analysis of traditional telephone and voice-over-Internet protocol (VoIP) systems. Electronics and the Environment, 2004. 2004 IEEE International Symposium. 10 - 13 May 2004. On page(s): 106 - 111.
14. Ethereal network protocol analyzer. Website 2007, (http://www.ethereal.com)
15. T. Walsh and D. Kuhn. Challenges in securing voice over IP. Security & Privacy Magazine, IEEE. Volume 3. Issue 3, on page(s): 44 - 49. May - June 2005.
16. B. Alfonsi. Alliance Addresses VoIP Security. Security & Privacy Magazine, IEEE. Volume 3, Issue 4, on page(s): 8. July - Aug 2005.
17. D. Estacion. Potential Security Problem looms for users of PC-based VoIP Products. IEEE Canadian Review. Feb 2005.
18. G. Hoglund, G. McGraw. Exploiting Software: How to break code. 2004.
19. Skype: A peer-to-peer Internet telephony network. Website 2007. (http://www.skype.com)
20. E. Aire, B. Maharaj, L. Linde. Implementation considerations in a SIP based secure voice over IP network. Africon 2004, 7th AFRICON Conference in Africa. 2004. Volume 1, on page(s): 167 – 172.
21. J. Kurose, K. Ross. Computer Networking, A top-down approach featuring the Internet, third edition. 2005.
22. C. Hung, C. Tan, L. Chuang, W. Lee. The implementation of the communication framework of SIP and MGCP in VoIP applications. Networks, 2002. ICON 2002. 10th IEEE International Conference. 27-30 Aug 2002. On page(s): 449 – 454.
23. N. Miglani. VoIP-An implementation and survey. 2001.

24. S. Shankar, J. del Prado, P. Wienert. Optimal packing of VoIP calls in an IEEE 802.111 a/e WLAN in the presence of QOS constraints and channel error. Global Telecommunications Conference, 2004. GLOBECOM'04. IEEE. 29 Nov - 3 Dec 2004. Volume 5, on page(s): 2974 – 2980.
25. S. Garg, M. Kappes. Can I add a VoIP call? Communications, 2003. ICC'03. IEEE International Conference. 11 - 15 May 2003. Volume 2, on page(s): 779 – 783.
26. W. Liew, Q. Li. A multiplex-multicast scheme that improves system capacity of voice-over-ip on wireless LAN by 100%. Computers and Communications, 2004. Proceedings. ISCC 2004. Ninth International Symposium. 28 June - 1 July 2004. Volume 1, on page(s): 472 - 477.
27. F. Dressler. Advantages of VoIP in the German research network. High Speed Networks and Multimedia Communications 5th IEEE International conference. 2002. On Page(s): 56 – 60.
28. H. Chong and Mathews. Comparative analysis of traditional telephone and voice-over-Internet protocol (VoIP) systems. Electronics and the Environment, 2004. Conference Record. 2004 IEEE International Symposium. 10-13 May 2004. On page(s): 106 – 111.
29. R. Hao, D. Lee, R. Sinha, N. Griffeth. Integrated system interoperability testing with applications to VoIP. Networking, IEEE/ACM Transactions. Oct 2004.Volume 12, Issue 5. On page(s): 823 – 836.
30. H. Xiao, P. Zarrella. Quality effects of wireless VoIP using security solutions. Military communications conference, 2004. Milcom 2004. IEEE. 31 Oct – 3 Nov 2004. Volume 3, on page(s) 1352 – 1357.
31. C. McKay and F. Masuda. Empirical Studies of wireless VoIP Speech Quality in the presence of Bluetooth Interference. Electromagnetic Compatibility, 2003 IEEE International Symposium. 18 - 22 Aug 2003. Volume 1, on page(s): 269 – 272.

32. B. Massad and S. Holthaus. Embedding call monitoring in VoIP designs. Website 2002. (http://www.commsdesign.com/article/printableArticle.jhtml?articleID=16504616

33. Alan Clark. Improving VoIP call quality with Embedded Monitoring. Website 2001. (http://www.commsdesign.com/showArticle.jhtml;jsessionid=UUO2KAJXYOT54QSNDLOSKH0CJUNN2JVN?articleID=16503492)

34. S. Garg, M. Kappes. An experimental study of throughput for UDP and VoIP traffic in IEEE 802.11b networks. Wireless Communications and Networking, 2003. WCNC 2003. 2003 IEEE. 16 – 20 March 2003. Volume 3, on page(s): 1748 – 1753.

35. H. Furuya, S. Nomoto, H. Yamada, N. Fukumoto, and F. Sugaya. Experimental investigation of the relationship between IP network performances and speech quality of VoIP. Telecommunications, 2003. ICT 2003. 10th International conference. 23 Feb – 1 March 2003. Volume 1, on page(s): 543 – 552.

36. VoIPtroubleshooter Website 2007. http://www.voiptroubleshooter.com/problems/jitterbuffer.html

37. C. Partridge, R. Hiden. RFC 1151. Version 2 of the Reliable Data Protocol (RDP).

38. D. Velten, R. Hinden, and J. Sax. RFC 908. Version 1 of the Reliable Data Protocol.

39. 4i2i, Website 2007. An Introduction to Reed-Solomon codes: principle, architecture and implementation. (http://www.4i2i.com/reed_solomon_codes.htm).

APPENDIX

APPENDIX A. RTP stream generator

```
##########################################################################
#                                                                        #
# LEGAL NOTICE                                                           #
#                                                                        #
# Ixia does not warrant that the functions contained in this             #
# Software will meet the user's requirements or that the                 #
# Software will be without omissions or error-free.                      #
#                                                                        #
# IN NO EVENT SHALL IXIA BE LIABLE FOR ANY DAMAGES RESULTING             #
# FROM OR ARISING OUT OF THE USE OF, OR THE INABILITY TO USE             #
# THE SOFTWARE OR ANY PART THEREOF, INCLUDING BUT NOT LIMITED            #
# TO ANY LOST PROFITS, LOST BUSINESS, LOST OR DAMAGED DATA OR            #
# SOFTWARE OR ANY INDIRECT, INCIDENTAL, PUNITIVE OR CONSEQUENTIAL        #
# DAMAGES, EVEN IF IXIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH      #
# DAMAGES IN ADVANCE                                                     #
#                                                                        #
# THE ORIGINAL VERSION OF THIS PROGRAM WAS MODIFIED BY                   #
# ALEX RIBADENEIRA TO SEND RTP PACKETS AT 20 MSEC INTERVALS WITH         #
# G.711 CODEC                                                            #
#                                                                        #
##########################################################################

package req IxTclHal
ixInitialize 192.168.0.10


proc makeEndPoints {portList ipList gwList prefixList}
   {
   set intf 0

   foreach port $portList
      {
      scan $port "%d %d %d" ch ca po
      interfaceTable select $ch $ca $po
      interfaceTable clearAllInterfaces
      interfaceEntry clearAllItems addressTypeIpV6
      interfaceEntry clearAllItems addressTypeIpV4
      ipAddressTable       setDefault
      ipAddressTable       set $ch $ca $po
      }

   foreach port $portList ip $ipList gw $gwList prefix $prefixList
      {
      scan $port "%d %d %d" ch ca po
      set mac [format "00 %02x %02x %02x 00 00" $ch $ca $po]
      set mac [join [lreplace $mac 4 5 [join [word2HexList $intf]]]]
      interfaceTable select $ch $ca $po
      ipAddressTable       setDefault

      interfaceIpV4  setDefault
      interfaceIpV4  config         -gatewayIpAddress       $gw
      interfaceIpV4  config         -maskWidth              $prefix
      interfaceIpV4  config         -ipAddress              $ip
      interfaceEntry addItem        addressTypeIpV4
```

```
      interfaceEntry setDefault
      interfaceEntry config         -enable                 true
      interfaceEntry config         -macAddress             $mac
      interfaceTable addInterface

      ipAddressTableItem setDefault
      ipAddressTableItem config    -fromIpAddress           $ip
      ipAddressTableItem config    -fromMacAddress          $mac
      ipAddressTableItem config    -numAddresses            1
      ipAddressTableItem config    -mappingOption           oneIpToOneMAC
      ipAddressTableItem config    -overrideDefaultGateway  true
      ipAddressTableItem config    -enableUseNetwork        true
      ipAddressTableItem config    -netMask                 $prefix
      ipAddressTableItem config    -gatewayIpAddress        $gw
      ipAddressTableItem config    -enableUseNetwork        true
      ipAddressTable     addItem
      ipAddressTable     set $ch $ca $po

      protocolServer setDefault
      protocolServer config         -enableArpResponse   true
      protocolServer config         -enablePingResponse  true
      protocolServer set            $ch $ca $po
      incr intf
      }

   set pl [lsort -unique $portList]
   ixWriteConfigToHardware pl
   ixTransmitArpRequest    pl
   }


proc initPortList {portList}
   {
   set pl [lsort -unique $portList]
   foreach port $pl
      {
      initPort $port
      }
   ixWritePortsToHardware pl
   after 1000
   ixCheckLinkState     pl
   }


proc initPort {port}
   {
   scan $port "%d %d %d" ch ca po
   port config -transmitMode            portTxModeAdvancedScheduler
   port config -receiveMode             portPacketGroup
   port config -autonegotiate           true
   port config -advertise1000FullDuplex true
   port config -advertise100FullDuplex  true
   port config -advertise100HalfDuplex  true
   port config -advertise10FullDuplex   true
   port config -advertise10HalfDuplex   true
   port set $ch $ca $po
   packetGroup setDefault
   packetGroup config -signatureOffset  50
   packetGroup config -groupIdOffset    54
   packetGroup setRx  $ch $ca $po
   interfaceTable select $ch $ca $po
   interfaceTable clearAllInterfaces
   interfaceEntry clearAllItems  addressTypeIpV6
```

```
   interfaceEntry clearAllItems  addressTypeIpV4
   arpServer setDefault
   arpServer config -retries            20
   arpServer config -mode               2
   arpServer config -rate               100
   arpServer config -requestRepeatCount 10
   arpServer set    $ch $ca $po
   port reset $ch $ca $po
   initPortData $port
   }

proc initPortData {port}
   {
   global portTable
   foreach codec [getCodecs]
      {
      set idx [join [list [join $port ,] $codec] ,]
      catch {unset portTable($idx)}
      set portTable($idx) [list 0 {}]
      }
   }


proc addCallData {port codec src dst udpPort num}
   {
   global portTable
   set idx [join [list [join $port ,] $codec] ,]
   if {![info exists portTable($idx)}
      {
      return
      }
   incr portTable($idx) $num
   set data [lindex $portTable($idx) 1]
   lappend $data [list $src $dst $udpPort $num]
   }


proc makeVoipStreams {port codec srcIpList dstIpList udpPortSource udpPortList}
   {
   global codecData
   set noCalls [llength $dstIpList]
   if {[llength $srcIpList] != [llength  $udpPortList]}
      {
      return 1
      }
   if {$noCalls != [llength $srcIpList]}
      {
      return 1
      }
   scan $port "%d %d %d" ch ca po
   set strm 1
   set found 0

   while {![stream get $ch $ca $po $strm]}
      {
      if {[stream cget -name] == [lindex $codecData($codec) 0]}
         {
         set found 1
         break
         }
      incr strm
      }
```

```
   if {$found}
      {
      udf get 1
      set tmpsrcIpList [udf cget -valueList]
      udf get 2
      set tmpdstIpList [udf cget -valueList]
      udf get 3
      set tmpudpPortList [udf cget -valueList]
      set  srcIpList   [ip2HexList $srcIpList]
      set  dstIpList   [ip2HexList $dstIpList]
      set  udpPortList [word2HexList $udpPortList]

      foreach srcIp $srcIpList dstIp $dstIpList udpPort $udpPortList
         {
         lappend tmpsrcIpList   $srcIp
         lappend tmpdstIpList   $dstIp
         lappend tmpudpPortList $udpPort
         }

      set  srcIpList   $tmpsrcIpList
      set  dstIpList   $tmpdstIpList
      set  udpPortList $tmpudpPortList
      set noCalls [llength $dstIpList]
      udp      setDefault
      udp      config      -sourcePort       $udpPortSource
      udp      config      -destPort         0
      udp      set         $ch $ca $po
      udf      setDefault
      udf      config     -enable            true
      udf      config     -counterMode       udfValueListMode
      udf      config     -offset            26
      udf      config     -countertype       c32
      udf      config     -valueList         $srcIpList
      udf      set        1
      udf      setDefault
      udf      config     -enable            true
      udf      config     -counterMode       udfValueListMode
      udf      config     -offset            30
      udf      config     -countertype       c32
      udf      config     -valueList         $dstIpList
      udf      set        2
      udf      setDefault
      udf      config     -enable            true
      udf      config     -counterMode       udfValueListMode
      udf      config     -offset            36
      udf      config     -countertype       c16
      udf      config     -valueList         $udpPortList
      udf      set        3

   } else {
      stream   setDefault
      stream   config      -sa               [format "00 AD E0 %02x %02x %02x"
$ch $ca $po]
      stream   config      -daRepeatCounter  daArp
      stream   config      -fir              true
      protocol setDefault
      protocol config      -name             ipV4
      protocol config      -appName          0
      protocol config      -ethernetType     ethernetII
      ip       setDefault
      ip       config      -ttl              63
      ip       config      -ipProtocol       udp
      ip       config      -sourceIpAddr     [lindex $srcIpList 0]
```

```
      ip       config      -destIpAddr       [lindex $dstIpList 0]
      ip       set         $ch $ca $po
      udp      setDefault
      udp      config      -sourcePort       $udpPortSource
      udp      config      -destPort         0
      udp      set         $ch $ca $po
      udf      setDefault
      udf      config     -enable            true
      udf      config     -counterMode       udfValueListMode
      udf      config     -offset            26
      udf      config     -countertype       c32
      udf      config     -valueList         [ip2HexList $srcIpList]
      udf      set        1
      udf      setDefault
      udf      config     -enable            true
      udf      config     -counterMode       udfValueListMode
      udf      config     -offset            30
      udf      config     -countertype       c32
      udf      config     -valueList         [ip2HexList $dstIpList]
      udf      set        2
      udf      setDefault
      udf      config     -enable            true
      udf      config     -counterMode       udfValueListMode
      udf      config     -offset            36
      udf      config     -countertype       c16
      udf      config     -valueList         [word2HexList $udpPortList]
      udf      set        3
   }

   if {[setCodecData $ch $ca $po $codec $noCalls]}
      {
      return 1
      }

   stream   set         $ch $ca $po $strm
   packetGroup setDefault
   packetGroup config -signatureOffset 50
   packetGroup config -signature       {08 71 18 05}
   packetGroup config -insertSignature true
   packetGroup config -groupIdOffset   54
   packetGroup config -groupId         0
   packetGroup setTx $ch $ca $po $strm
   stream   write       $ch $ca $po $strm
}

proc setCodecData {ch ca po {codec G.711u} {noCalls 1}}
   {
   global codecData

   if {![info exists codecData($codec)]}
      {
      return 1
      }

   stream config -name                    [lindex $codecData($codec) 0]
   stream config -framesize               [lindex $codecData($codec) 1]
   stream config -pattern        [lrange  [lindex $codecData($codec) 4] 0 end]
   stream config -patternType             nonRepeat
   stream config -dataPattern             userpattern
   stream config -rateMode                usePercentRate
   stream config -percentPacketRate       [expr $noCalls *
[calculatePercentMaxRate 1 $ca $po [lindex $codecData($codec) 2] [lindex
$codecData($codec) 1]]]
```

```
   udf    setDefault
   udf    config -enable                 true
   udf    config -offset                 46
   udf    config -countertype            c32
   udf    config -counterMode            udfNestedCounterMode
   udf    config -continuousCount        true
   udf    config -updown                 uuuu
   udf    config -initval                {00 00 00 00}
   udf    config -repeat                 1
   udf    config -step                   [lindex $codecData($codec) 3]
   udf    config -innerRepeat            $noCalls
   udf    config -innerStep              [lindex $codecData($codec) 3]
   udf    config -innerLoop              $noCalls
   udf    set              5
   udf    setDefault
   udf    config  -enable                true
   udf    config  -offset                44
   udf    config  -countertype           c16
   udf    config -counterMode            udfNestedCounterMode
   udf    config -continuousCount        true
   udf    config -updown                 uuuu
   udf    config -initval                {00 01}
   udf    config -repeat                 1
   udf    config -step                   1
   udf    config -innerRepeat            $noCalls
   udf    config -innerStep              1
   udf    config -innerLoop              1
   udf    set              4
   return 0
}

proc initCodecTable {}
   {
   global codecData
   catch {unset codecData}
   set codecData(G.711a) [list {G.711a (64kbps)} 218   50  160  {

      80 08 03 9C 00 02 0C A8 7E DF F9 D0 00 02 02 60

      64 C4 CA 58 48 0B B6 D6 E0 8B 60 2E 5B 65 B1 3D

      BA C1 53 9C 0B DE E0 83 C7 92 DC 7C E9 66 F9 0A

      4C 6B E6 19 46 47 62 63 5B 12 FF 3E 1F 55 33 3B

      0A 11 2A 66 C8 1C 4B 43 32 30 E6 8B 6F D4 BF 24

      2E 84 FC 98 32 64 3A 9B 3F E0 41 84 E4 AE 6F 7E

      9E 74 73 73 F9 4F B8 EC 6B E1 C7 47 12 FD 45 5C

      2B A2 DC 7A 11 60 51 A0 79 41 B0 C8 DB 1B AF 87

      7B 18 5D 1F 4E 0E 1B A9 60 60 72 2F 71 47 95 EC

      71 09 4F 68 93 31 80 F5 F3 73 48 91 67 43 E3 AD

      55 68 32 6B BB 4B EE 8D C2 8B E1 00
      } ]
   }


proc getCodecs {}
```

```
   {
   global codecData
   return [lsort -dictionary [array names codecData]]
   }


proc getCodecData {codec {param all}}
   {
   global codecData
   switch $param
      {
      all      {return $codecData($codec)}
      name     {return [lindex $codecData($codec) 0]}
      size     {return [lindex $codecData($codec) 1]}
      rate     {return [lindex $codecData($codec) 2]}
      tstamp   {return [lindex $codecData($codec) 3]}
      data     {return [lindex $codecData($codec) 4]}
      }
   return {}
   }


proc ip2HexList {addr}
   {
   set tmp {}
   foreach ip $addr
      {
      scan $ip "%d.%d.%d.%d" a b c d
      lappend tmp [format "%02x %02x %02x %02x" $a $b $c $d]
      }
   return $tmp
   }


proc word2HexList {val}
   {
   set tmp {}
   foreach item $val
      {
      lappend tmp [format "%02x %02x" [expr ($item >> 8) & 0xff] [expr $item &
0xff]]
      }
   return $tmp
   }


proc makeEndPointPairs {pl baseIp baseGw prefix numPairs}
   {
   set portList   {}
   set ipList     {}
   set gwList     {}
   set prefixList {}

   set netInc     [expr 1 << (32-$prefix)]
   set maxHosts   [expr $netInc - 3]
   if {$numPairs > $maxHosts}
      {
      return 1
      }

   foreach port $pl
      {
      set ip $baseIp
```

```
      for {set a 0} {$a < $numPairs} {incr a}
         {
         lappend portList     $port
         lappend ipList       $ip
         lappend gwList       $baseGw
         lappend prefixList   $prefix
         set ip [bin2Ip [mpexpr 1 + [ip2Bin $ip]]]
         }

      set baseIp [bin2Ip [mpexpr [ip2Bin $baseIp] + $netInc]]
      set baseGw [bin2Ip [mpexpr [ip2Bin $baseGw] + $netInc]]
      }

   makeEndPoints $portList $ipList $gwList $prefixList
   }


proc makeVoipTraffic {port codec srcIp dstIp numPairs callsPerPair srcudpPort
udpPort}
   {
   set srcList {}
   set dstList {}
   set udpList {}
   for {set pair 0} {$pair < $numPairs} {incr pair}
      {
      set uPort $udpPort
      for {set call 0} {$call < $callsPerPair} {incr call}
         {
         lappend srcList $srcIp
         lappend dstList $dstIp
         lappend udpList $uPort
         incr uPort
         }
      set srcIp  [bin2Ip [mpexpr 1 + [ip2Bin $srcIp]]]
      set dstIp  [bin2Ip [mpexpr 1 + [ip2Bin $dstIp]]]
      }
   makeVoipStreams $port $codec $srcList $dstList $srcudpPort $udpList
   }


proc bin2Ip {val}
   {
   set a [mpexpr ($val & 0xff000000) >> 24]
   set b [mpexpr ($val & 0x00ff0000) >> 16]
   set c [mpexpr ($val & 0x0000ff00) >>  8]
   set d [mpexpr ($val & 0x000000ff) >>  0]
   return [format "%d.%d.%d.%d" $a $b $c $d]
   }


proc ip2Bin {addr}
   {
   scan $addr "%d.%d.%d.%d" a b c d
   return [mpexpr (($a << 24) | ($b << 16) | ($c << 8) | $d)]
   }


proc clearAllStreams {portList}
   {
   foreach port $portList
      {
      scan $port "%d %d %d" ch ca po
      port reset $ch $ca $po
```

```
      }
   }


proc getLatencyInfo {port buckets}
   {
   set retVal {}
   scan $port "%d %d %d" ch ca po
   packetGroupStats get $ch $ca $po 0 $buckets
   for {set group 0} {$group < $buckets} {incr group}
      {
      if {[packetGroupStats getGroup $group]}
         {
         continue;
         }
      set tot [packetGroupStats cget -totalFrames]
      set avg [packetGroupStats cget -averageLatency]
      set max [packetGroupStats cget -maxLatency]
      set min [packetGroupStats cget -minLatency]
      lappend retVal [list $tot $min $max $avg]
      }
   return $retVal
   }


proc getLat {port}
   {
   set latInfo [getLatencyInfo $port 6000]
   set idx 1
   foreach bucket $latInfo
      {
      ixPuts [format "%4d - %d" $idx [lindex $bucket 0]]
      incr idx
      }
   }


proc whatRates {port}
   {
   scan $port "%d %d %d" ch ca po
   ixPuts [format "%-25s %s"  Codec "Percent Line Rate for One call"]

   foreach codec [getCodecs]
      {
      ixPuts [format "%-25s %s" [getCodecData $codec name] [expr
[calculatePercentMaxRate 1 $ca $po [getCodecData $codec rate] [getCodecData
$codec size]  ]]]
      }
   }


#main
set pl            {{1 2 1} {1 2 2}}
set baseGw1       10.9.0.1
set baseIp1       10.9.100.1

#10.9.0.1    peer 10.9.100.1
#10.9.0.2    peer 10.9.100.2

set prefix        16
#10.9.x.x/16 belong to the same network. DUT is a switch

set numPairs      1
```

```
set callsPerPair  1

initCodecTable

initPortList      $pl

makeEndPointPairs $pl $baseIp1 $baseGw1 $prefix $numPairs

clearAllStreams   $pl

makeVoipTraffic {1 2 1}  G.711a    $baseIp1 $baseGw1   $numPairs $callsPerPair
1025 2000
makeVoipTraffic {1 2 2}  G.711a    $baseGw1 $baseIp1   $numPairs $callsPerPair
2000 1025

whatRates {1 2 1}
whatRates {1 2 2}

ixStartPacketGroups pl;after 6100;getLat {1 2 1}
ixPuts "Sending: $numPairs call"
```

APPENDIX B. Mean opinion score calculator

```
<html>

<head>
<title>E-model calculator</title>
</head>

<body>

<script type="text/javascript">
document.writeln("<b>Mean opinion score calculator</b><br>");
document.writeln("Author:    Alexander F. Ribadeneira<br>");
document.writeln("Reference: ITU-T G.107, ITU-T G.113, RFC 3550, RFC
3551");

function calculate()
{

/*********************************************************************
*
//Default values for the parameters of the E-model according ITU-T
G.107
var SLR  =   8;         //Send Loudness Rating
var RLR  =   2;         //Receive Loudness Rating
var STMR =  15;         //Sidetone Masking Rating
var LSTR =  18;         //Listener Sidetone Rating
var Ds   =   3;         //D-Value of Telephone, Send Side
var Dr   =   3;         //D-Value of Telephone Receive Side
var TELR =  65;         //Talker Echo Loudness Rating
var WEPL = 110;         //Weighted Echo Path Loss
var T    =   0;         //Mean one-way Delay of the Echo Path
var Tr   =   0;         //Round-trip Delay in a 4-wire Loop
var Ta   =   0;         //Absolute Delay in echo-free Connections
var qdu  =   1;         //Number of Quantization Distortion Units
var Ie   =   0;         //Equipment Impairment Factor
var Bpl  =   1;         //Packet-loss Robustness Factor
var Ppl  =   0;         //Random Packet-loss Probability
var Nc   = -70;         //Circuit Noise referred to 0 dBr-point
var Nfor = -64;         //Noise Floor at the Receive Side
var Ps   =  35;         //Room Noise at the Send Side
var Pr   =  35;         //Room Noise at the Receive Side
var A    =   0;         //Advantage Factor
*********************************************************************
***/
var SLR  =   parseFloat(document.getElementById("_SLR").value);
var RLR  =   parseFloat(document.getElementById("_RLR").value);
var STMR =   parseFloat(document.getElementById("_STMR").value);
var Ds   =   parseFloat(document.getElementById("_Ds").value);
var Dr   =   parseFloat(document.getElementById("_Dr").value);
var TELR =   parseFloat(document.getElementById("_TELR").value);
var WEPL =   parseFloat(document.getElementById("_WEPL").value);
var T    =   parseFloat(document.getElementById("_T").value);
var Tr   =   parseFloat(document.getElementById("_Tr").value);
var Ta   =   parseFloat(document.getElementById("_Ta").value);
```

```
var qdu  =   parseFloat(document.getElementById("_qdu").value);
var Ie   =   parseFloat(document.getElementById("_Ie").value);
var Bpl  =   parseFloat(document.getElementById("_Bpl").value);
var Ppl  =   parseFloat(document.getElementById("_Ppl").value);
var Nc   =   parseFloat(document.getElementById("_Nc").value);
var Nfor =   parseFloat(document.getElementById("_Nfor").value);
var Ps   =   parseFloat(document.getElementById("_Ps").value);
var Pr   =   parseFloat(document.getElementById("_Pr").value);
var A    =   parseFloat(document.getElementById("_A").value);

//Calculate LSTR
var LSTR =   STMR + Dr;
document.getElementById("_LSTR").value = LSTR;

//Calculate OLR
var OLR = SLR + RLR;

//Calculate Nfo
var Nfo = Nfor + RLR;

//Calculate Pre
var Pre = Pr + 10 * log10 (1 + Math.pow (10,(10-LSTR)/10));

//Calculate Nor
var Nor = RLR - 121 + Pre + 0.008 * (Pre - 35) * (Pre - 35);

//Calculate Nos
var Nos = Ps - SLR - Ds - 100 + 0.004 * (Ps - OLR - Ds -14) * (Ps - OLR
- Ds -14);

//Calculate No
var No  = 10 * log10 ( Math.pow(10,(Nc/10)) + Math.pow(10,(Nos/10)) +
Math.pow(10,(Nor/10)) + Math.pow(10,(Nfo/10)) );

//Calculate Ro
var Ro  = 15 - 1.5 * ( SLR + No);
document.getElementById("_Ro").value = Math.round(Ro*10)/10;

//Calculate Xolr
var Xolr = OLR + 0.2 * (64 + No - RLR);

//Calculate Iolr
var Iolr = 20 * (        Math.pow((1 + Math.pow((Xolr/8),8)),0.125)
-    Xolr/8   );

//Calculate STMRo
var STMRo = -10 * log10 ( Math.pow (10,(-STMR/10)) + Math.exp(-T/4) *
Math.pow (10,(-TELR/10)));

//Calculate Ist
var Ist = 12 * Math.pow (    1 + Math.pow(  (STMRo -13)/6     , 8)
, 0.125);
Ist -=    28 * Math.pow (    1 + Math.pow(   (STMRo +1)/19.4 , 35)
,(1/35));
Ist +=   -13 * Math.pow (    1 + Math.pow(   (STMRo -3)/33   , 13)
,(1/13)) + 29;
```

```
//Calculate Q
var Q = 37 - 15 * log10 (qdu);

//Calculate G
var G = 1.07 + 0.258 * Q + 0.0602 * Q * Q;

//Calculate Z
var Z = 46/30 - G/40;

//Calculate Y
var Y = (Ro - 100)/15 + 46/8.4 - G/9;

//Calculate Iq
var Iq = 15 * log10 ( 1 + Math.pow(10,Y) + Math.pow(10,Z) );

//Calculate Is = Iolr +  Ist + Iq
var Is = Iolr + Ist + Iq;
document.getElementById("_Is").value = Math.round(Is*10)/10;

//Calculate TERV
var TERV = TELR - 40 * log10 ( (1 + T/10) / ( 1 + T/150 ) ) + 6 *
Math.exp( -0.3 * T * T );

if  ( STMR < 9 )
{
TERV = TERV + Ist/2;
}

//Calculate Re
var Re  = 80 + 2.5 * (TERV - 14);

//Calculate Roe
var Roe = -1.5 * (No - RLR);

//Calculate Idte
var Idte = (  (Roe-Re)/2 + Math.sqrt ( (Roe-Re)*(Roe-Re)/4 + 100 ) - 1)
*   ( 1 - Math.exp(-T) );

if ( STMR > 20)
{
Idte = Math.sqrt (Idte*Idte + Ist*Ist);
}

if ( T < 1 )
{
Idte = 0;
}

//Calculate  Rle
var Rle = 10.5 * ( WEPL + 7 ) * Math.pow((Tr + 1),(-0.25));

//Calculate Idle
var Idle = (Ro-Rle)/2 + Math.sqrt ((Ro-Rle)*(Ro-Rle)/4 + 169);

//Calculate Idd
var Idd;
if ( Ta > 100 )
```

```
{
var X = log10(Ta/100) / log10(2);
Idd = 25 * (     Math.pow ( (1 + Math.pow (X,6)) , (1/6))  -  3 *
Math.pow ( (1 + Math.pow (X/3,6)) , (1/6))     + 2 );
}

else
{
Idd = 0;
}

//Calculate Id
var Id = Idte + Idle + Idd;
document.getElementById("_Id").value = Math.round(Id*10)/10;


//Calculate Ie_eff
var Ie_eff = Ie + ( 95 - Ie) * Ppl / (Ppl + Bpl);
document.getElementById("_Ie_eff").value = Math.round(Ie_eff*10)/10;

//Calculate R
var R = Ro - Is - Id - Ie_eff + A;
document.getElementById("_R").value = Math.round(R*10)/10;

//Calculate MOS
var MOS;

if (R<0)
{
MOS = 1;
}

else if (R>0 && R<100)
{
MOS = 1 + 0.035 * R + R * ( R - 60 ) * ( 100 - R ) * 7 * Math.pow(10,-
6);
}

else if (R>100)
{
MOS = 4.5;
}

document.getElementById("_MOS").value = Math.round(MOS*10)/10;

}

function log10(x)
{
return (Math.log(x)/Math.log(10))
}

function divide_string()
{
var Ie_Bpl=document.getElementById("_Codec").value;
var Ie_Bpl_array = Ie_Bpl.split("|");
document.getElementById("_Ie").value = Ie_Bpl_array[0];
```

```
document.getElementById("_Bpl").value = Ie_Bpl_array[1];
}

</script>

<br><br>

<DIV ALIGN=CENTER>
<table border="0">
<tr>
<td><h4 align="center">R</h4></td>
<td><h4 align="center">=</h4></td>
<td><h4 align="center">Ro</h4></td>
<td><h4 align="center">-</h4></td>
<td><h4 align="center">Is</h4></td>
<td><h4 align="center">-</h4></td>
<td><h4 align="center">Id</h4></td>
<td><h4 align="center">-</h4></td>
<td><h4 align="center">Ie_eff</h4></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><h4 align="center">MOS</h4></td>
</tr>

<tr>
<td><input type="text" id="_R"  maxlength="4" size="2"></td>
<td>          </td>
<td><input type="text" id="_Ro" maxlength="4" size="2"></td>
<td>          </td>
<td><input type="text" id="_Is" maxlength="4" size="2"></td>
<td>          </td>
<td><input type="text" id="_Id" maxlength="4" size="2"></td>
<td>          </td>
<td><input type="text" id="_Ie_eff" maxlength="4" size="2"></td>
<td>          </td>
<td>          </td>
<td>          </td>
<td>          </td>
<td>          </td>
<td><input type="text" id="_MOS" maxlength="4" size="2"></td>
</tr>
</table>
</DIV>

<br><br>


<DIV ALIGN=CENTER>
<table border="0">
<tr>
<td><input type="submit" value="reset" onClick="_SLR.value=8"></td>
<td><input type="text" id="_SLR" value="8" maxlength="4" size="6"></td>
<td><b>Sender Loudness Rating (SLR) dB</b></td>
<td>     </td>
```

```
<td><input type="submit" value="reset" onClick="_T.value=0"></td>
<td><input type="text" id="_T" value="0" maxlength="4" size="6"></td>
<td><b>Mean one-way Delay of the echo Path (T) msec</b></td>
</tr>

<tr>
<td><input type="submit" value="reset" onClick="_RLR.value=2"></td>
<td><input type="text" id="_RLR" value="2" maxlength="4" size="6"></td>
<td><b>Receive Loudness Rating (RLR) dB</b></td>
<td>     </td>
<td><input type="submit" value="reset" onClick="_Tr.value=0"></td>
<td><input type="text" id="_Tr" value="0" maxlength="4" size="6"></td>
<td><b>Round Trip Delay in a 4-wire Loop (Tr) msec</b></td>
</tr>

<tr>
<td><input type="submit" value="reset" onClick="_STMR.value=15"></td>
<td><input type="text" id="_STMR" value="15" maxlength="4"
size="6"></td>
<td><b>Sidetone Masking Rating (STMR) dB</b></td>
<td>     </td>
<td><input type="submit" value="reset" onClick="_Ta.value=0"></td>
<td><input type="text" id="_Ta" value="0" maxlength="4" size="6"></td>
<td><b>Absolute Delay in echo-free Connections (Ta) msec</b></td>
</tr>

<tr>
<td></td>
<td><input type="text" id="_LSTR" value="18" maxlength="4" size="6"
disabled ></td>
<td><b>Listener Sidetone Rating (LSTR) dB = STMR + Dr</b></td>
<td>     </td>
<td><input type="submit" value="reset" onClick="_qdu.value=1"></td>
<td><input type="text" id="_qdu" value="1" maxlength="4" size="6"></td>
<td><b>Number of quantization Distortion Units (qdu)</b></td>
</tr>

<tr>
<td><input type="submit" value="reset" onClick="_Ds.value=3"></td>
<td><input type="text" id="_Ds" value="3" maxlength="4" size="6"></td>
<td><b>D-Value of Telephone, Send Side (Ds)</b></td>
<td>     </td>
<td></td>
<td><input type="text" id="_Bpl" value="1" maxlength="4" size="6"
style="background-color:#9CA5F5" style="color:#000000"></td>
<td><b>Packet-loss Robustness Factor (Bpl)</b></td>
</tr>

<tr>
<td><input type="submit" value="reset" onClick="_Dr.value=3"></td>
<td><input type="text" id="_Dr" value="3" maxlength="4" size="6"></td>
<td><b>D-Value of Telephone, Receive Side (Dr)</b></td>
<td>     </td>
<td><input type="submit" value="reset" onClick="_Ppl.value=0"></td>
<td><input type="text" id="_Ppl" value="0" maxlength="4" size="6"></td>
<td><b>Random packet loss probability (Ppl) %</b></td>
</tr>
```

```
<tr>
<td><input type="submit" value="reset" onClick="_TELR.value=65"></td>
<td><input type="text" id="_TELR" value="65" maxlength="4"
size="6"></td>
<td><b>Talker Echo Loudness Rating (TELR) dB</b></td>
<td>     </td>
<td><input type="submit" value="reset" onClick="_Nc.value=-70"></td>
<td><input type="text" id="_Nc" value="-70" maxlength="4"
size="6"></td>
<td><b>Circuit Noise referred to 0 dBr-point (Nc) dBm0p</b></td>
</tr>

<tr>
<td><input type="submit" value="reset" onClick="_WEPL.value=110"></td>
<td><input type="text" id="_WEPL" value="110" maxlength="4"
size="6"></td>
<td><b>Weighted Echo Path Loss (WEPL) dB</b></td>
<td>     </td>
<td><input type="submit" value="reset" onClick="_Nfor.value=-64"></td>
<td><input type="text" id="_Nfor" value="-64" maxlength="4"
size="6"></td>
<td><b>Noise Floor at the Receive Side (Nfor) dBmp</b></td>
</tr>

<tr>
<td></td>
<td><input type="text" id="_Ie" value="0" maxlength="4" size="6"
style="background-color:#9CA5F5" style="color:#000000"></td>
<td><b>Equipment Impairment Factor (Ie)</b></td>
<td>     </td>
<td><input type="submit" value="reset" onClick="_Ps.value=35"></td>
<td><input type="text" id="_Ps" value="35" maxlength="4" size="6"></td>
<td><b>Room noise at the Send Side (Ps) dB(A)</b></td>
</tr>

<tr>
<td></td>
<td><b>Codec Type</b></td>
<!-- For Ie|Bpl values for the codecs described below refer to ITU
G.113 -->
<td><SELECT id="_Codec" onChange="divide_string()">
<OPTION VALUE="0|1">
<OPTION VALUE="0|4.3">G.711
<OPTION VALUE="0|25.1">G.711 PLC
<OPTION VALUE="15|16.1">G.723.1 6.3k
<OPTION VALUE="11|19.0">G.729A
</SELECT></td>

<td>     </td>
<td><input type="submit" value="reset" onClick="_Pr.value=35"></td>
<td><input type="text" id="_Pr" value="35" maxlength="4" size="6"></td>
<td><b>Room Noise at the Receive Side (Pr) dB(A)</b></td>
</tr>

<tr>
<td></td>
```

```
<td></td>
<td></td>
<td>     </td>
<td><input type="submit" value="reset" onClick="_A.value=0"></td>
<td><input type="text" id="_A" value="0" maxlength="4" size="6"></td>
<td><b>Advantage Factor (A)</b></td>
</tr>

</table>

<br><br>

<td><input type="submit" name="Update" value="Update"
onClick="calculate()"></td>
</DIV>

</body>
</html>
```