An Adaptive Playout Algorithm with Delay Spike Detection for Real-Time VoIP

Aziz Shallwani

Department of Electrical & Computer Engineering
McGill University
Montreal, Canada

October 2003

A thesis submitted to the Faculty of Graduate and Postdoctoral Studies in partial fulfillment of the requirements for the degree of Master of Engineering.

To Apajan and Bhaijan, for your constant love and support.

Abstract

As the Internet is a best-effort delivery network, audio packets may be delayed or lost en route to the receiver due to network congestion. To compensate for the variation in network delay, audio applications buffer received packets before playing them out. Basic algorithms adjust the packet playout time during periods of silence such that all packets within a talkspurt are equally delayed. Another approach is to scale individual voice packets using a dynamic time-scale modification technique based on the WSOLA algorithm.

In this work, an adaptive playout algorithm based on the normalized least mean square algorithm, is improved by introducing a spike-detection mode to rapidly adjust to delay spikes. Simulations on Internet traces show that the enhanced bi-modal playout algorithm improves performance by reducing both the average delay and the loss rate as compared to the original algorithm.

Résumé

Puisque l'Internet est un réseau où la qualité de service n'est pas garantie, des paquets audio peuvent être retardés ou perdus en chemin vers le destinataire à cause de la congestion du réseau. Pour palier aux variations du délai réseau, les applications audio accumulent les paquets reçus avant de les jouer. Les algorithmes simples ajustent le délai de lecture des paquets pendant les périodes de silence de sorte que tous les paquets dans une section de conversation sont également retardés. Une autre approche est de redimensionner les paquets individuellement avec une technique de modification temporelle basée sur l'algorithme WSOLA.

Dans ce travail, un algorithme de lecture adaptatif basé sur la méthode normalisée du plus petit carré moyen est amélioré en ajoutant un mode détectant les pics de délai réseau pour rapidement ajuster la prédiction du délai durant ces pics. Des simulations réalisées à partir de traces récupérées sur Internet montrent que le modèle de prédiction bi-mode amélioré réduit à la fois le délai moyen et le taux de perte par rapport à l'algorithme original.

Acknowledgments

Firstly, I would like to acknowledge my supervisor, Prof. Peter Kabal, for his valuable guidance and continuous encouragement throughout my graduate studies at McGill University. Prof. Kabal has always been very patient and supportive. I would like to thank the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Canadian Space Agency (CSA) for the financial support enabling me to carry out my research.

I would like to extend my thanks to Julien Vanier. Julien implemented the packet-based WSOLA algorithm that was used in my research. Moreover, I would like to thank Julien for his help with the translation of the thesis abstract. I would also like to acknowledge Noureen Naaz Alladin, Naazish Alladin, Mark Klein, and Apurv Shah for their help in gathering the network delay traces.

I am eternally grateful to my colleagues and friends in the Telecommunications & Signal Processing Laboratory; Tarun Agarwal, Chris Cave, Ricky Der, Ahmed Usman Khalid, Mark Klein, Youssouf Ould-Cheikh-Mouhamedou, Benoit Pelletier, Kamal Deep Singh Sahambi, Paxton Smith and others. I would also like to thank Kamakshi Advani, Simon Gee, Tony Leung, Arif Mohammadi, and Zafrin Nurmohamed. Without their close friendship and support, I would not have been able to complete this work. They have always been there for me through both good and bad times.

Finally, I am sincerely indebted to my parents, my sisters, and my grandparents for their love and understanding. They have encouraged me, supported me, loved me, and given me more than I could ever ask for. Thank you.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Voice over Internet Protocol (VoIP) has quickly become one of the fastest-growing technologies in the world today. A relatively new field, VoIP is the transmission of packetized voice over packet-based IP networks such as the Internet. VoIP traffic grew by almost 900% from 1998 to 1999, and is projected to grow another 5000% between 1999 and 2004 [1].

Traditionally, communications networks have been split along two lines. The legacy, *circuit-switched* Public Switched Telephone Network (PSTN) was designed for real-time voice conversations between users. Subsequently, the PSTN has been adapted for the growing needs of data communications (e.g., modem technologies, ISDN) [2]. In contrast, *packet-based* networks such as the Internet, were created for the efficient transportation of data. Data has overtaken voice as the primary traffic on many networks built for voice [3], and data traffic continues to grow faster than voice traffic. More recently, voice traffic is also being transported over packet networks.

The integration of voice and data networks onto a single packet-based data network infrastructure is driving the rapid growth in Internet telephony or VoIP. The convergence of voice and data onto a single network allows network resources to be used efficiently. The transmission of real-time voice and multimedia over data networks is supported by the wide-scale deployment of high-performance digital signal processors (DSPs) [4].

1.1 PSTN vs. VoIP

The PSTN and the Internet carry voice calls quite differently. The PSTN sets up an exclusive, dedicated connection with a fixed bandwidth from the calling party to the called

number. The voice signal is transmitted as a synchronous stream of binary data over the connection. The dedicated circuit/connection is then maintained for the duration of the call.

The Internet, on the other hand, is a connectionless packet-based network. The voice signal is packetized and encoded into data packets. Each packet is then routed independently through the network. Packets can incur variable delay due to congestion at routers within the network, and may even be discarded by a router to alleviate excessive congestion. Furthermore, packets in a voice stream may be routed over different paths in the network, resulting in a non-sequential arrival at the receiver [1].

1.2 Voice over IP Drivers

There has been a growing interest in VoIP within the communications industry for a number of reasons, as described in [4].

1.2.1 Integration of Voice and Data

The integration of real-time audio, video, and data will allow network operators to offer and support new multimedia applications as well as other unified services.

1.2.2 Bandwidth Consolidation

The integration of voice and data allows for bandwidth consolidation, by using the data network more efficiently. In a traditional PSTN telephone call, bandwidth is allocated to the customer for the duration of the phone call. Typically, 50% of the speech pattern in most voice conversations is silence. Traditional voice networks waste precious bandwidth carrying the periods of silence, whereas data networks do not send the silence and instead the bandwidth is made available to other users who need it at that instant. As well by using analog-to-digital converters (ADC) and compression algorithms, the speech bit rate can be reduced from 64 kbit/s in the current PSTN network to a bit rate between 4.8 and 8 kbit/s in a data network [4].

1.2.3 Lower Tariffs

The greatest driver of VoIP is the extremely low costs associated with placing calls over the Internet. VoIP systems bypass the PSTN's toll services and use the Internet backbone to transmit voice calls. The costly long distance charges found in the PSTN are avoided, and the lower costs of sending voice packets through the Internet are incurred instead.

1.2.4 Universal Presence of IP

Personal home computers and workstations in industry currently use the Internet and the Internet Protocol for data transfer. Thus, a convenient and extensive network infrastructure is already in place for voice packet transmission.

1.2.5 Maturation of Technologies

High-performance digital signal processors (DSPs) are used by codecs (voice coders and decoders) and high-speed modems. The real-time processors are highly optimized to process digital signals. In VoIP systems, the analog voice signal is digitized and encoded before being transmitted over the network. DSPs are now mass-produced and have become relatively inexpensive, allowing IP telephony to become a feasible and practical alternative to the PSTN.

1.2.6 Shift to Packet-Based Networks

Current trends show that there is a shift from circuit-switched networks to packet-based data networks. Data traffic currently exceeds voice traffic, and is expected to grow at a faster rate than voice traffic [3]. By packetizing voice and transmitting it over the Internet, IP telephony allows for the replacing of voice networks with data networks. With the convergence of voice and data networks, only the IP-based packet network needs to be supported. Maintenance costs are generally lower for packet-based routers or switches than it is for circuit-based switches [1].

1.3 VoIP Challenges

While VoIP provides substantial benefits, a number of challenges must be met before VoIP can be successfully deployed on a wider scale. To become a credible alternative to the PSTN, VoIP must offer the same reliability and voice quality. High end-to-end voice quality in packet transmission networks depends principally on the following factors:

- Signal fidelity, i.e., choice of speech codec (**co**der/**dec**oder) used
- End-to-end delay and variation in the delay (*jitter*)
- Packet loss

1.3.1 Speech Signal Fidelity

The analog voice signal is digitized and encoded at the transmitter. Speech codecs may be used to compress the digital audio signal and reduce the bit rate for transmission. The digitization and compression of a speech signal may produce a noticeable degradation in the voice quality, depending on the speech coding algorithm used. The signal quality can deteriorate further under conditions of packet loss. However, the perceived reduction in speech quality depends on the effectiveness of any associated packet loss concealment (PLC) algorithms.

1.3.2 End-to-End Delay and Jitter

The end-to-end delay (also called "latency") is the time between the generation of speech at the transmitter and its playout at the receiver. End-to-end delays beyond 400 ms are irritating to telephone users and impair interactivity in real-time conversations [5]. Talker echo also becomes more noticeable and annoying as delay increases.

The end-to-end delay in VoIP comprises the *processing delay*, the *network delay*, and the *buffering delay*. The *processing delay* consists of a) the coding delay and b) the packetization delay associated with the speech codec. The coding delay is the processing time needed to encode and decode the signal. The packetization delay consists of the duration of signal contained in a voice packet, typically between 10 and 40 ms. The processing delay also includes delays due to other DSP features such as echo cancellation, noise reduction, and packet loss concealment.

The best-effort nature of packet networks results in packets experiencing variable delay and loss while propagating through the network. The delay incurred by voice packets while traversing the packet network from transmitter to receiver is called the *network delay*. The variation in network delay, referred to as *jitter*, must be smoothed out before packets are played out at the destination. A playout buffer is used by the receiver to store packets and remove the variation in delay before playing them out. The additional delay imposed by the playout (or jitter) buffer is the *buffering delay*.

1.3.3 Packet Loss

The unreliability of IP networks can lead to packets being lost occasionally. Network congestion results in long packet queues at network routers. Routers may choose to deliberately drop some packets to reduce the congestion. Packets may also be lost at network nodes due to buffer overflow.

To provide reliable transmission of data in an IP network, a retransmission scheme is provided by the transport layer. The scheme retransmits any packets for which an acknowledgement was not received from the destination. Retransmission schemes cannot be used for real-time voice transmission, as the end-to-end packet delays would be too large and inhibit conversation.

Similarly, packets incurring high network delay may arrive after their scheduled playout time. Although the packets reach the destination, they arrive too late to be played out by the receiver and are considered to be 'lost'. The number of late packets can be reduced by increasing the buffering delay. However, a larger jitter buffer increases the overall end-to-end delay.

The lost packets create gaps in the reconstructed audio signal, which can result in clicks, muting or unintelligible speech. Speech codecs use packet loss concealment (PLC) algorithms to reduce the degradation caused by packet loss.

1.4 Playout Buffer Algorithms

Adaptive playout buffer algorithms react to changing network conditions by dynamically adjusting the end-to-end delay. Since audio packets are generated at regular intervals, the received packets must be played out in a periodic manner. A typical real-time voice

conversation consists of talkspurts (when the user is talking), and periods of silence (when the user is listening).

Playout delay adjustments made during periods of silence are less likely to be perceived by users. The playout delay is adjusted on a per-talkspurt basis by stretching or compressing the silence between talkspurts. The basic playout approach [6] is to estimate the end-to-end delay and use it to set the playout time of the first packet in a talkspurt. Subsequent packets in the talkspurt will have the same total end-to-end delay.

In a recent approach to adaptive playout, playout delay adjustment is performed within talkspurts [7]. Individual voice packets are time-scaled such that they are played out just in time for the predicted arrival time of the next packet. A time-scale modification technique, based on the Waveform Similarity Overlap-Add (WSOLA) algorithm [8], can be used to time-scale voice packets within talkspurts while preserving the voice pitch. The degradation in perceptual quality due to time-scaling has been found to be inaudible [7]. Dynamically adjusting the playout delay during talkspurts, improves overall performance by allowing for a reduction in the end-to-end delay while maintaining low packet loss. The main approaches to estimation of playout delay are described in the following subsection.

1.4.1 Approaches to Playout Delay Estimation

There are three main types of playout delay estimation algorithms: *autoregressive (AR) estimate-based* algorithms, *statistically-based* algorithms, and *adaptive filter-based* algorithms.

The basic playout algorithm proposed in [6] uses an autoregressive (AR) estimate to calculate the network delay and jitter. The end-to-end delay is then computed to be the sum of the network delay estimate and a safety buffer term, so that only a small fraction of packets will arrive too late to be played out by the receiver. The safety buffer term depends on the jitter estimate.

Statistically-based approaches use the statistics of past delays to compute the current playout delay. The network delays for past packets are retained and the playout delay is selected such that a chosen percentage of packets should arrive in time [7, 9]. Only the delays of the past talkspurt are used in [10]. Another approach assembles all previous delay values in a histogram and applies an aging procedure to gradually reduce the impact of older packet delays [11].

Instead of reacting to network fluctuations, the adaptive filter-based approach to adaptive playout attempts to predict the network delay [12]. Adaptive filtering algorithms aim to minimize the expected mean square error between the actual network delay and the estimate [13]. Previous delays are passed through a finite-impulse response (FIR) filter to compute the current estimate. The mean square error is then used to adjust the tap weights of the adaptive filter. The normalized least mean square (NLMS) algorithm is used for the adaptive predictor in [12].

The playout buffer algorithms considered here are not robust enough to adapt delay estimates in the presence of a delay spike. A delay spike is characterized by the sudden onset of a large increase in network delay. Although subsequent packets usually experience declining network delays, the delay values are still quite large.

A spike-detection algorithm was first developed by Ramjee *et al.* [6] to adapt to such spikes. The playout algorithm switches to an impulse or spike mode when a delay spike is detected. In spike mode, the delay estimate depends only on the most recent delay values. When the delays fall back to average values, the algorithm reverts to normal mode and the delay estimate is computed using the AR estimate-based approach. Other playout buffer algorithms also modify their delay estimates during spikes [7, 9, 10].

1.5 Thesis Contribution

Playout buffer algorithms which adjust the playout delay on a per-packet basis, can be improved further with more accurate estimates of the network delay. The NLMS playout algorithm [12] adjusts the playout delay based on a prediction of the network delay. Since the NLMS playout algorithm does not detect delay spikes, the predicted network delays become very large during spikes.

This thesis proposes an enhancement to the NLMS algorithm. A spike mode is incorporated to rapidly adjust to network delay spikes. As the onset of a delay spike is followed by declining network delays, the safety buffer term can be reduced during spikes, thereby reducing both the buffering delay and overall end-to-end packet delay. As well, since a more accurate estimate of the delay is obtained, packets received during a delay spike will not be stretched or compressed as much, thereby reducing any degradation associated with time-scaling of voice packets.

1.6 Thesis Organization

The remainder of this thesis is as follows:

Chapter 2 presents an overview on Voice over IP (VoIP). The chapter first describes the network protocol used in IP networks, the Internet Protocol (IP). The various components of a VoIP transmission systems are then presented and the major protocols used in VoIP are also reviewed.

Chapter 3 presents the idea of jitter reduction through buffering received packets. The chapter begins with an explanation of the purpose of a playout (or jitter) buffer. The main types of adaptive playout buffer algorithms are then revisited. Playout delay adjustment is usually performed during periods of silence between talkspurts. A recent approach to adaptive playout algorithms, which adjusts the playout delay on a per-packet basis, is also described. Lastly, the drawbacks of the current NLMS playout algorithm are detailed and an enhanced NLMS predictor, the E-NLMS playout algorithm, is proposed.

Chapter 4 evaluates the performance of the proposed E-NLMS playout algorithm. The method used to evaluate the E-NLMS algorithm is described first. Various issues regarding end-to-end network delay traces, as well as the process used to collect them, are also detailed. Finally, the simulations and experimental results evaluating the E-NLMS algorithm for the assorted network delay traces are presented and discussed.

Lastly, Chapter 5 summarizes and presents the conclusions of the thesis. Suggestions for future research directions are also given.

Chapter 2

Voice Over IP (VoIP)

The desire to integrate voice and data networks has fuelled interest in the area of Voice over IP (VoIP). Traditionally, packet-based networks such as the Internet have been designed for data communications and not for real-time voice transmission. However, VoIP provides considerable advantages over the Public Switched Telephone Network (PSTN). The migration from circuit-switched networks to packet-based Internet Protocol (IP) networks is expected to continue in the future.

Chapter 2 gives a general background on Voice over IP. A brief introduction to the Internet Protocol (IP) is given in Section 2.1. The essential components of a VoIP transmission system are described in Section 2.2. Lastly, some of the more popular VoIP protocols are reviewed in Section 2.3.

2.1 Internet Protocol (IP)

The TCP/IP reference model is the name given to the family of communications protocols used to support applications in the Internet. The TCP/IP reference model is divided into layers, and each layer is responsible for a different aspect of the communication. The layers of the model are listed and described below [14].

1. The *link layer* (also called the *network interface layer*) consists of the device driver in the operating system as well as the network interface card on the PC. The link layer handles all the hardware details of physically interfacing with the actual network cable. Examples of link layer protocols are Ethernet and Token Ring.

2. The *network layer* (also called the *internet layer*) handles the movement of data packets between different hosts in the network. The Internet Protocol (IP) [15] is the protocol used at the network layer in the TCP/IP reference model.

3. The *transport layer* manages and provides the flow of data between the two end hosts for the application layer. There are two transport protocols in the TCP/IP reference model, the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP).

4. The *application layer* handles the high-level details of particular applications. Some common TCP/IP applications provided include [14, 16]:

   - Telnet for remote login
   - The File Transfer Protocol (FTP)
   - The Simple Mail Protocol (SMTP) for e-mail
   - The Hyper Text Transfer Protocol (HTTP) for browsing websites on the World Wide Web (WWW)

2.1.1 Overview of IP

IP is a connectionless protocol. Traffic is exchanged between two computers without any prior call setup. The computers can share a connection at the transport layer. However, at the network layer where IP resides, packets are individually transmitted and routed through the network from sender to receiver.

IP is a best-effort, datagram-type protocol. No quality of service is guaranteed and no error recovery is provided. There are no flow-control or retransmission mechanisms. Packets or datagrams may be lost, duplicated, or even arrive out of order due to possibly different paths taken through the network. For example, there is a maximum queue length at an IP gateway. If an IP packet arrives and the maximum queue length is surpassed, the buffers at the router will overflow, and IP packets will be dropped. It is the responsibility of the transport layer protocol to recover from such problems [4].

IP hides the underlying subnetwork from the user. This property is advantageous because different types of networks can use IP. For example, the IP protocol is supported over a variety of media, such as Asynchronous Transfer Mode (ATM), frame relay, dedicated

lines, Integrated Services Digital Network (ISDN), Ethernet, and Digital Subscriber Line (DSL) [14].

The low-level characteristics of IP make it suitable for real-time voice transmission. IP is reasonably simple to install and because of its connectionless design, it is quite robust [4]. The universal presence of IP allows for widespread acceptance and use of Internet telephony.

IP Datagram

The structure of an IP datagram is depicted in Fig. 2.1. The IP datagram consists of a header and the data. The header has a 20-byte fixed part, and a variable length options part. The IP datagram is transmitted in big-endian order, i.e., the most significant bit (MSB) first. The various fields in the IP datagram header are briefly described here.

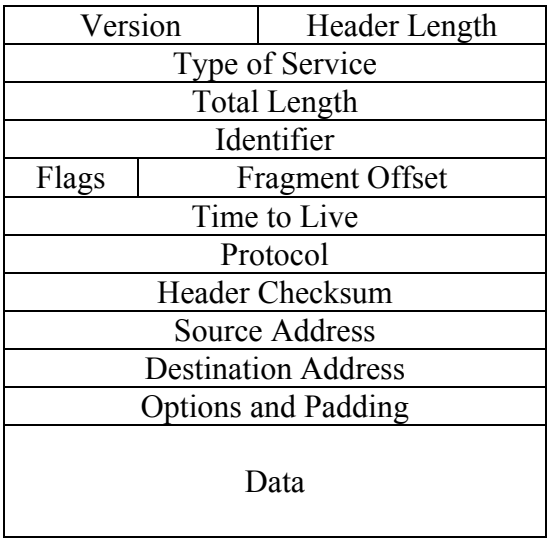


Fig. 2.1 The IP datagram [4]

Version (4 bits): This field identifies the version of the IP protocol. The majority of current IP-based networks use version 4 of the IP protocol. While IP version 6 has been developed, the protocol has not yet been deployed on a wide scale.

Header Length (4 bits): This field is set to the length of the IP datagram header. The length is measured in 32-bit words.

Type of Service (8 bits): This field is used to describe the quality of service requested by the sender for this IP packet or protocol data unit (PDU). The first three bits, bits 0–2 contain a *Precedence* value indicating the priority of the PDU. The precedence options are

Routine (000), Priority (001), Immediate (010), Flash (011), Flash Override (100), Critical (101), Internetwork Control (110), and Network Control (111). The next three bits are used for other services. Bit 3 is the *Delay bit (D bit)*; if the bit is set to 0, the PDU can be delayed and if it is set to 1, low delay is requested. The next bit is the *Throughput bit (T bit)*. The bit is set to 0 for normal throughput, and high throughput within the network is requested by setting the bit to 1. Bit 5 is the *Reliability bit (R bit)*. The bit is set to 0 for normal reliability and to 1 to request high reliability. The last two bits, bits 6 and 7 are not currently used and are reserved for future use.

Total Length (16 bits): The length of the total IP datagram is given by the total length field. The total length is measured in 8-bit octets and includes the length of the header and the data. The maximum possible length of an IP datagram is 65,536 (2^{16}) octets. Such long datagrams are impractical for most hosts and networks. However, IP does require that all underlying networks be able to handle IP datagrams up to 576 octets in total length.

The IP protocol uses three fields (*Identifier*, *Flags*, and *Fragment Offset*) in the header to manage the fragmentation and reassembly of IP datagrams.

Identifier (16 bits): All fragments of an initial IP PDU have the same unique identifier. The identifier is used to aid in reassembling a fragmented IP PDU.

Flags (3 bits): The flags field indicates whether the PDU can be fragmented. The first bit is reserved and set to 0. The second bit, is the *Don't Fragment bit (DF bit)*. If the bit is set to 0, the IP PDU can be segmented, and if it is set to 1, the PDU cannot be segmented. The third bit is the *More Fragments bit (MF bit)*. The MF bit is set to 1 if there are more segments, and it is set to 0 if there are no more segments. If the present segment is the last segment or the only segment, then the MF bit is set to 0.

Fragment Offset (13 bits) The fragment offset field specifies the relative position of the fragment in the original datagram. The value represents the number of 64-bit blocks (excluding header octets) that are contained in earlier fragments. The first fragment has an offset value of 0.

Time to Live (8 bits): The TTL field indicates the maximum time in seconds that the PDU is allowed to remain in circulation on the Internet. While the TTL is measured in time, it is interpreted as the maximum number of hops. Each gateway that processes the IP PDU must decrement the value by at least one. When the value of the field reaches zero, the packet is assumed to be undeliverable and is discarded.

Protocol (8 bits): This field identifies the transport layer protocol above the IP layer,

that is to receive the datagram at the destination.

Header Checksum (16 bits): The header checksum is used to detect any bit errors that may have occurred in the header. The checksum is computed on the header only and not on the user data stream. The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words in the header [15]. For the purposes of computing the checksum, the value of the header checksum field is zero.

Source Address (32 bits): The source address field contains the IP address of the sender of this PDU. An IP node is identified by its IP address, which consists of both a network address and a host address. The network address specifies the subnetwork (or subnet), while the host address identifies the particular node on the subnet.

Destination Address (32 bits) The IP address of the receiver of the IP datagram is stored in the destination address field.

Options (variable length): This field defines additional services. The field is optional and is not used in every IP datagram. Most implementations use this field for network management and diagnostics.

2.1.2 Transport Layer Protocols

As IP is an unreliable, connectionless, best-effort delivery service, it is up to the transport layer protocols to ensure the reliable transport of packets between two hosts. The two main transport layer protocols are the Transmission Control Protocol (TCP) [17] and the User Datagram Protocol (UDP) [18].

Transmission Control Protocol (TCP)

TCP provides full-duplex, acknowledged, and flow-controlled service to upper-layer applications. The data is moved in a continuous, unstructured byte stream, where bytes are identified by sequence numbers [3].

TCP is a connection-oriented protocol providing reliable delivery of IP packets. A connection is set-up between two hosts before any data is exchanged. The destination sends an acknowledgement to the sender upon receiving a TCP packet. If a packet acknowledgement is not received by the sender, the source retransmits the packet and the transmission rate is dynamically reduced. After the sender receives an acknowledgement for an outstanding packet, the sender slides the packet window along the byte stream and sends another

packet. This flow control mechanism is known as a *sliding window* protocol [3].

Within the signalling portion of VoIP, TCP is used to ensure the reliability of the call setup. However, as end-to-end delay is critical for voice transmission, the TCP protocol is not used for the actual transmission of voice packets in a VoIP call.

User Datagram Protocol (UDP)

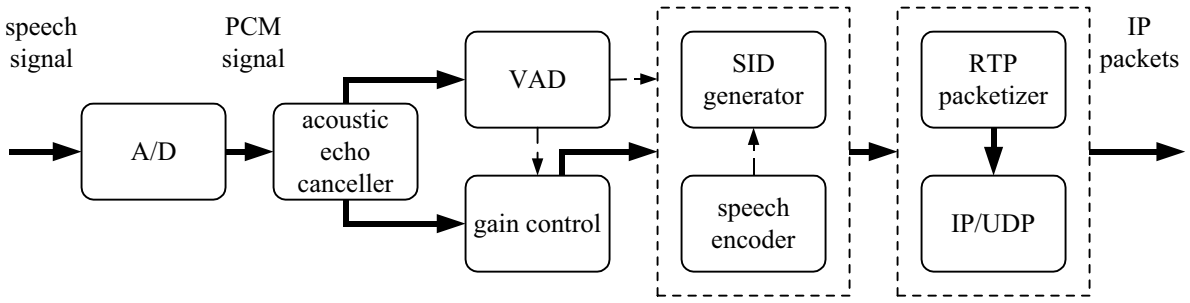
The UDP protocol on the other hand, is a connectionless protocol and does not provide sequencing or acknowledgements. UDP is a simpler protocol than TCP and is useful in situations where the full services of TCP are not needed. UDP has a smaller header than TCP, resulting in minimal overhead. Since UDP has no reliability, no flow control, nor error-recovery, it serves principally as a multiplexer/demultiplexer for the receiving and sending of IP traffic [4].

UDP is used in VoIP to carry the actual voice traffic. TCP is not used because the retransmission of lost voice packets would result in extra delay. If TCP were utilized for VoIP, the end-to-end delay incurred while waiting for acknowledgements and retransmission, would render the voice quality unacceptable. With VoIP and other real-time applications, reducing the latency is more important than ensuring the reliable delivery of every voice packet [3].

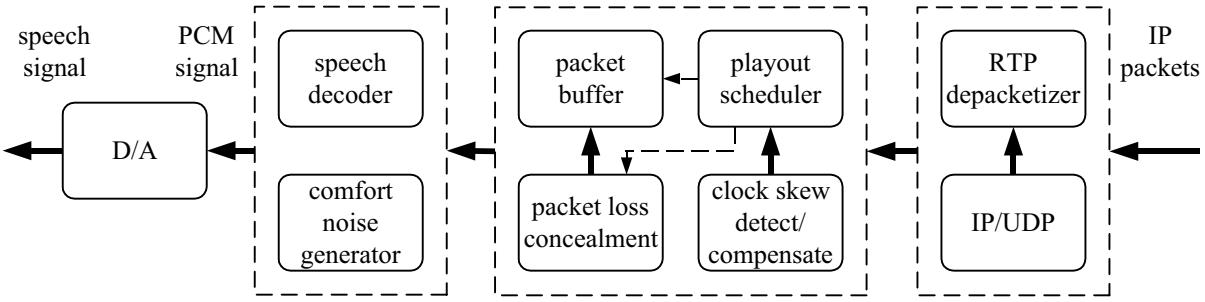
2.2 VoIP Transmission Systems

The transmission of real-time voice from one point to another in a VoIP system consists of multiple steps [1]. At the sender, the continuous analog voice stream is digitized by being periodically sampled and encoded. The digital signal is then processed to remove line echoes. A voice activity detector is used to identify periods of silence. During silence, the sender can either not transmit any packets or it can reduce the bit rate. The signal is framed using a speech codec (**co**der/**dec**oder). The speech codec may also compress the signal. The voice frame is then packetized for transport over the network. A Real-Time Transport Protocol (RTP) packet is created by adding a 12-byte header to the compressed voice frame. The RTP packet is then encapsulated into a UDP packet at the transport layer and into an IP packet at the network layer. The IP packet is then sent onto the Internet, where the packet is routed to its destination.

Since packets may be lost or delayed through the network, a playout buffer is used at the receiver to remove network delay jitter and to store packets until their scheduled playout time. The receiver extracts the compressed voice frame, decompresses it and converts it back to analog form for playout. Packets that do not arrive at the destination or arrive too late to be played out in time, are assumed to be 'lost'. Concealment algorithms can be used to compensate for lost packets. A generic VoIP transmission system is illustrated in Fig. 2.2. The various components of a VoIP transmission system are described here.



(a) A VoIP transmitter.



(b) A VoIP receiver.

Fig. 2.2 A generic VoIP transmission system [19]

2.2.1 Speech Codecs

Speech coding is used in telecommunications to achieve efficient transmission of a speech signal from one host to another. Human speech contains energy up to 10 kHz in frequency, but high intelligibility is maintained even if components up to only 3.4 kHz are kept. In the traditional PSTN network, a speech signal is sampled at 8 kHz and then bandpass filtered

from 300–3400 Hz [20]. The signal is sampled with 12 or 13 bits/sample using pulse code modulation (PCM). The log PCM technique assigns more levels to lower-amplitude signals and reduces the bit rate to 8 bits/sample, resulting in an overall bit rate of 64 kbps.

Speech coders can be classified into three categories. The simplest type of speech coders are *waveform coders*. Waveform coders attempt to code and reconstruct the speech waveform on a sample-by-sample basis (e.g., PCM, adaptive differential PCM (ADPCM)). Time-domain waveform coders take advantage of waveform redundancies (e.g. periodicity and slowly varying intensity) to allow data compression, while spectral-domain waveform coders exploit the nonuniform distribution of speech information across frequencies [20]. More complex speech coders known as *source coders* or *parametric vocoders* (*voice coders*) attempt to model the human speech production mechanisms rather than the actual speech waveforms. The vocoders aim to characterize the vocal tract shape and excitation source using a small set of parameters (e.g., linear predictive coding (LPC)). Lastly, *hybrid coders* combine the features of both waveform coders and vocoders (e.g., code excited linear prediction (CELP)).

Voice Coding Standards

The International Telecommunication Union - Telecom Standardization (ITU-T) has standardized CELP, Algebraic-Code-Excited Linear-Prediction (ACELP), Multipulse Maximum Likelihood Quantization (MP-MLQ), PCM, and ADPCM in its G-series recommendations. The most popular voice coding standards for telephony and packet voice include [3]:

- G.711: Describes the 64 kbps PCM coding technique used in the PSTN. G.711 voice samples are already in the correct format for delivery within the PSTN.
- G.726: Describes ADPCM coding at 16, 24, 32 and 40 kbps. ADPCM packets can be interchanged between packet networks and the PSTN, provided that the latter supports ADPCM.
- G.728: Describes a 16 kbps low-delay variation of CELP voice compression.
- G.729: Describes a CELP compression standard enabling voice to be coded at 8 kbps.
- G.723.1: Describes a compression technique used to compress speech or other audio. The higher 6.3 kbps bit rate is based on MP-MLQ technology and provides greater

quality. The lower 5.3 kbps bit rate is based on ACELP and provides good quality.

Speech codecs are judged based on subjective measurements of voice quality. Standard objective quality measurements, such as total harmonic distortion and signal-to-noise ratios, do not correspond well to a human's perception of voice quality. A subjective measure used to evaluate the quality of speech is the *mean opinion score (MOS)*. A number of users are asked to listen to a speech sample and asked to rate it on a 5-point scale from 1 (Unsatisfactory) to 5 (Excellent). The scores are then averaged to give the mean opinion score [21]. Table 2.1 summarizes the popular voice coding standards.

Table 2.1 Commonly used speech codecs [3]

Standard	Method	Bit Rate (kbps)	Delay (ms)	Quality (MOS)
G.711	PCM	64	0.125	4.1
G.726	ADPCM	32	0.125	3.85
G.728	LD-CELP	15	0.625	3.61
G.729	CS-ACELP	8	10	3.92
G.729A	CS-ACELP	8	10	3.7
G.723.1	MP-MLQ	6.3	30	3.9
G.723.1	ACELP	5.3	30	3.65

2.2.2 Voice Activity Detection

In a typical voice conversation, each participant will speak for 50% of the time and listen for 50% of the time. A voice activity detector (VAD) is used at the sender to differentiate between speech and silence. As will be described in the next section, VADs are used to implement silence suppression / discontinuous transmission (DTX) in VoIP transmitters.

A basic VAD functions by comparing the average signal energy to a noise threshold for each frame. Speech is detected if the signal energy is greater than the noise threshold. When the VAD detects a drop-off in signal energy, it waits a fixed amount of time before silence is declared. This amount of fixed time is known as *hangover* and is typically 200 ms [3].

An inherent problem with VADs is detecting when speech restarts. The beginning of a sentence may be cut off or clipped when transitioning from silence to speech. This phenomenon is referred to as *front-end clipping*. In general, front-end clipping > 50 ms is perceivable and has a negative effect on speech quality [19].

2.2.3 Silence Suppression / DTX

A major advantage of VoIP is that bandwidth can be efficiently allocated to users. Speech packets only need to be sent when a person is speaking. During periods of silence when a person is listening, the bit rate can be significantly reduced. This dual-rate transmission technique is known as *silence suppression* or *discontinuous transmission (DTX)*.

A VAD is used at the transmitter to detect the silence periods. Silence suppression schemes cease to transmit speech packets during periods of silence. In DTX mode, Silence Insertion Descriptor (SID) packets are sent at the beginning of, and intermittently throughout the silence period [19]. The SID frames are smaller than speech codec data frames, and contain parameters used to generate background noise. DTX is preferred over silence suppression as the parameters of the background noise are transmitted to the receiver. As well, DTX enables the sender and receiver to maintain synchronization.

2.2.4 Comfort Noise Generation

Silence suppression / DTX schemes result in the receiver having no packets for playout during periods of silence. If absolute silence is played out at the receivers, listeners may be confused into believing the connection has been lost. To overcome this problem, receivers employ comfort noise generation (CNG) whereby background noise is generated and played out at the destination during periods of silence. Receivers generate background noise based on parameters contained in the intermittent SID packets sent by the transmitter [22].

2.2.5 Echo Cancellation

Echo is the reflection of a signal through the network, with sufficient delay to be perceived by the user. An echo with a delay in excess of approximately 32 ms can be annoying to the speaker [22]. Echo cancellers are deployed in voice networks to reduce or eliminate echo. The echo cancellers estimate and subtract the echo estimate from the received signal. There are two types of echoes in voice communications: *hybrid* and *acoustic*.

Hybrid echo occurs in the PSTN network and is caused by a mismatch in impedance at the hybrid from the four-wire trunk to the two-wire local loop. The hybrid separates the send and receive paths in order to carry them on separate wires. As the separation of send and receive paths is not perfect, the receive signal is partially reflected onto the send path, and an echo is generated [22]. Echo cancellation is performed on the four-wire side of the

hybrid [2]. Hybrid echo does not occur in pure VoIP calls, but is present in PSTN-to-IP calls.

Acoustic echo normally occurs when a free-air microphone and speakers are used. The remote user's speech signal is played out through the speakers and is picked up by the microphone and transmitter back to the remote user. Adaptive filtering algorithms can be used for acoustic echo cancellation.

2.2.6 Playout Scheduling

VoIP transmitters generate packets at regular intervals and send them onto the network towards the receiver. The best-effort nature of connectionless packet-based networks such as the Internet results in packets incurring varying network delay due to different levels of congestion in the network. Typically, packets arrive at the receiver at irregular intervals. The variation in packet interarrival time is called *jitter*. Playout buffers are used at the destination to reduce jitter.

Playout buffers remove jitter by buffering the received packets for a short period of time before playing them out at scheduled intervals. Packets arriving after their scheduled playout time are late and considered to be 'lost'. Thus, there is a tradeoff between the packet loss rate and the playout delay. Adaptive playout buffering algorithms attempt to adjust the playout buffering delay according to current network conditions. The main types of adaptive playout algorithms are reviewed in Chapter 3. The E-NLMS playout buffering algorithm proposed in this thesis, is also presented in Chapter 3.

2.2.7 Packet Loss Concealment

Packet loss occurs in VoIP systems when packets do not reach the intended destination or they arrive after their scheduled playout time, and are too late to be played out at the receiver. Packet Loss Concealment (PLC) algorithms are used at the receiver to compensate for late and lost packets. Simple PLC algorithms use silence or noise substitution for lost packets. Using subsequent packets (when available) to interpolate and reconstruct lost packets has also been recently proposed [23].

Forward error correction (FEC) schemes are introduced to speech codecs to add redundancy [24, 25]. Packet loss is minimized by adjusting the bit rate as well as increasing the amount of redundancy during periods of high network congestion. A novel approach inte-

grates packet FEC into adaptive playout buffer algorithms [26]. Packet loss due to network congestion can be effectively compensated by using a combination of PLC algorithms and FEC schemes.

2.3 VoIP Protocols

While VoIP is a relatively new field, a number of application layer protocols have been developed for Internet telephony. The Real-Time Transport Protocol (RTP) [27] provides mechanisms for real-time delivery of voice, video, and data. The ITU-T H.323 [28] protocol suite and the Internet Engineering Task Force (IETF) Session Initiation Protocol (SIP) [29] standard are two of the major call signalling protocols in VoIP. The standards covering such topics as call-setup and tear-down procedures, socket/port numbers, and other call-control procedures. The MEGACO/H.248 protocol [30] has been developed to control the gateways between circuit-switched and packet-based networks. The RTP, H.323, SIP and MEGACO/H.248 protocols are described in this section.

2.3.1 Real-Time Transport Protocol (RTP)

Media transport in IP-based telephony is generally implemented with the Real-Time Transport Protocol (RTP) [27]. RTP provides end-to-end transport of real-time data, such as audio and video. An RTP packet consists of an RTP header and media payload. RTP packets are customarily transported over UDP/IP. RTP does not guarantee QoS, nor does it address the issue of resource reservation along the path of a connection [22]. However, the RTP header contains a sequence number so that receivers can detect the occurrence of lost packets and can present received packets in correct order. Additionally, RTP packets include the sender timestamp. The destination can use this timestamp to calculate network delay and jitter, as well as ensure synchronized playout.

Media encoding is explicitly identified in the RTP packet payload format with *RTP profiles*. The profile defines how a class of payloads (e.g., audio and video) is carried by RTP. The real-time transport of audio using different speech codecs is specified in the “RTP Profile for Audio and Video Conferences with Minimal Control” defined in RFC 1890 [31].

RTP Packet Header

The basic RTP header is small and only 12 bytes in length. Fig. 2.3 presents the basic RTP header. The various fields are briefly described [22].

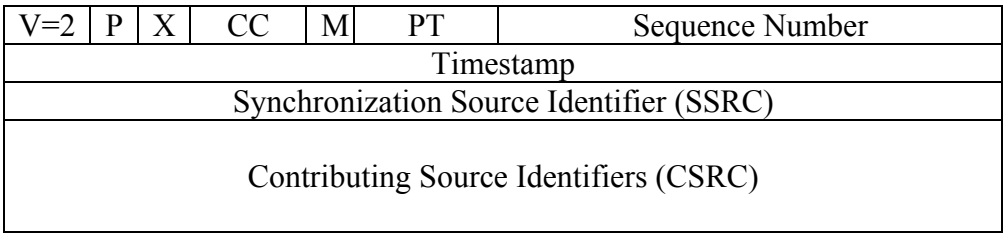


Fig. 2.3 Basic RTP packet header [22]

Version (V, 2 bits): This field identifies the version of RTP. The current version defined in RFC 1889 [27] is version 2.

Padding (P, 1 bit): If the padding bit is set to 1, the actual data content in the packet is less than the size of the packet. The packet must always be aligned on a 32 bit boundary. The last byte of the padding gives the number of bytes at the end of the packet that should be ignored.

Extension (X, 1 bit): The extension bit indicates whether the fixed RTP header is followed by an RTP extension. The RTP extension header is four bytes long and contains the type of the extension (2 bytes) and the length of the extension (2 bytes). The RTP extension header must be inserted after the last valid field in the standard RTP header. The last valid field is either the Synchronization Source Identifier (SSRC) field, or the last Contributing Source Identifier (CSRC) entry if there are any, as explained below.

CSRC Count (CC, 4 bits): The CSRC count contains the number of CSRC identifiers that follow the fixed header. For example, RTP mixers collect multiple media packets from multiple sources, combine them into a single packet, and forward the packet to a destination. The various sources whose packets have been combined are identified with a Contributing Source Identifier (CSRC). The CSRC count gives the number of such sources. In the standard case with only one source, the source is identified by the Synchronization Source Identifier (SSRC) and the CSRC Count field is set to zero.

Marker (M, 1 bit): The interpretation of the marker bit is defined by the RTP profile. The marker is intended to mark significant events such as frame boundaries. In the case of

real-time audio and video, the marker bit is used to indicate the beginning of a talkspurt, or the end of a video frame. Receivers can use this information to determine when to generate comfort noise at the receiver.

Payload Type (PT, 7 bits): This field identifies the format of the RTP payload. The RTP profile maps the payload type codes to payload formats which are then appropriately interpreted by the application. Multiplexing of different media streams in the same packet is impossible as the sender can only have one payload type per packet.

Sequence Number (16 bits): The sequence number increments by one for each RTP data packet sent. The sequence number can be used by the receiver to detect packet loss as well as reorder packets arriving out of order. The initial value of the sequence number is randomly generated.

Timestamp (32 bits): The timestamp is set when the first octet in the RTP data packet is sampled. The timestamp must be derived from a clock that increments monotonically and linearly in time to allow synchronization and jitter calculation. For speech, the timestamp is incremented by one for each sampling period. An RTP packet containing 20 ms of speech sampled at 8 kHz will have its timestamp incremented by $8000 \times 0.02 = 160$.

Synchronization Source Identifier (SSRC) (32 bits): The SSRC field identifies the synchronization source or sender. The value of the SSRC is chosen randomly. If two senders within the same RTP session have the same SSRC identifier, a simple mechanism in RTP is used to resolve the collision.

Contributing Source Identifiers (CSRC) (0 to 15 items, 32 bits each): The CSRC field contains the synchronization source identifiers for all the contributing sources for the payload contained in the packet. In general, the field is only used for packets that have been mixed. The number of sources is given by the *CSRC count* field. The CSRC identifiers are the original SSRCs of the packet sources, and are inserted by the RTP mixers. The CSRC field is used for correct source identification when payloads are played out at the destination endpoint.

RTP Control Protocol (RTCP)

The companion control protocol to RTP described in RFC 1889 [27] is the RTP Control Protocol (RTCP). RTCP packets are used to provide statistics about the quality of the session, user information, and time synchronization. RTCP measures network performance

by computing statistics such as packet delay, packet loss and jitter at the receiver. Using these statistics, endpoints can adapt to varying network conditions. Since RTCP packets add to network congestion, the bandwidth consumed by RTCP is set to 5% of the total bandwidth allocated for the session. Additionally, the mean interval between RTCP packets is set to be a minimum of 5 seconds [32].

RTP Header Compression

The RTP/UDP/IP headers are 12, 8, and 20 bytes, respectively, adding up to a 40-byte header per packet. The 40-byte header is twice the size of the payload when transmitting two packets coded using G.729; each G.729 packet corresponds to 10 ms of speech. The RTP/UDP/IP headers can be compressed using compressed RTP (cRTP) [33]. The cRTP compression scheme reduces the combined RTP/UDP/IP header from 40 bytes to 2 bytes when UDP checksums are not used, and to 4 bytes when they are used.

Although several header fields change between successive RTP/UDP/IP packets, the differences are often constant. By maintaining both the uncompressed header and the values of the first-order differences in the session state, cRTP needs to only send the second-order differences. The compressed RTP packet header is sent in place of the uncompressed packet header about 98% of the time. An uncompressed header must be sent periodically to ensure that both endpoints have the correct state. As well, if changes occur in a field that is usually constant, then the combined RTP/UDP/IP header cannot be compressed, and an uncompressed header is transmitted [3].

2.3.2 H.323

The H.323 standard [28] is an ITU-T specification for transmitting audio, video and data over an IP network, including the Internet. The H.323 standard covers call signalling and control, multimedia transport and control, and bandwidth control for point-to-point and multipoint conferences [3]. The H.323 standard consists of the protocols listed in Table 2.2.

H.323 Elements

The key components in an H.323 system are terminals, gateways, gatekeepers, and multipoint control units (MCUs). Fig. 2.4 illustrates the various elements in an H.323 system. The various elements are briefly described here.

Table 2.2 H.323 protocol suite [3]

Feature	Protocol
Call Signalling	H.225
Media Control	H.245
Audio Codecs	G.711, G.722, G.723.1, G.728, G.729
Video Codecs	H.261, H.263
Data Sharing	T.120
Media Transport	RTP/RTCP

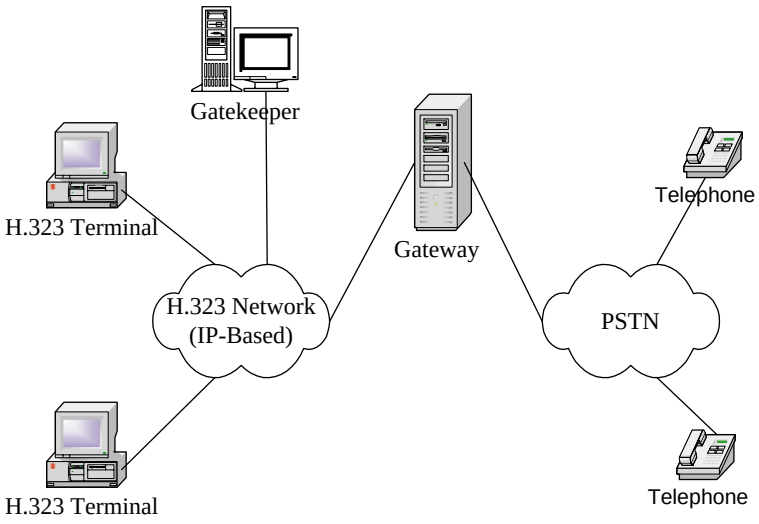


Fig. 2.4 H.323 network components [23]

Terminals: Terminals or endpoints provide point-to-point communications and multipoint conferences with other H.323 terminals. The terminal must support audio communications, and can optionally support video and data sharing.

Gateways: A gateway provides an interface between two different networks. The H.323 gateway connects an H.323 network and a non-H.323 network such as the PSTN. The gateway translates between audio, video, and data transmission formats as well as communication systems and protocols including call setup and release. Gateways are only needed for interconnection with non-H.323 networks, and therefore not required for communication between two H.323 terminals.

Gatekeepers: Gatekeepers provide pre-call and call-level control services to H.323 endpoints. Gatekeepers are optional. However, if a gatekeeper is present in an H.323 system, it must perform address translation, admissions control, bandwidth control and zone management. Optional gatekeeper functions include call control signalling, call authorization, bandwidth management, call management, gatekeeper management information and directory services [14]

Multipoint Control Units (MCUs): Multipoint control units are endpoints that support conferences between three or more endpoints. The MCU typically consists of a multipoint controller (MC) and one or more multipoint processors (MPs). The MC handles the control and signalling to support the conference while the MPs receive audio, video, and/or data streams, process them and distributes them to the endpoints participating in the multipoint conference.

H.323 Protocol Suite

The H.323 protocol suite consists of several protocols. The protocol suite supports call admissions, setup, status, release, media streams, and messages in H.323 systems. The protocols are supported by both reliable and unreliable packet delivery transport mechanisms over IP networks, as illustrated in Fig. 2.5.

The H.323 protocol suite consists of three main areas of control [3]:

- Registration, Admissions, and Status (RAS) Signalling: RAS signalling provides pre-call control in H.323 gatekeeper-based networks.
- Call Control Signalling (H.225): Based on ITU-T Recommendation H.225 [34], which specifies the use of Q.931 signalling messages, call control procedures are used to

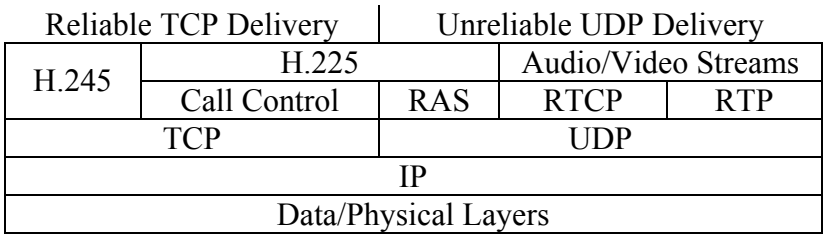


Fig. 2.5 H.323 protocol suite layers [3]

connect, maintain, and disconnect calls between endpoints.

- Media Control and Transport (H.245 and RTP/RTCP): H.245 [35] governs the transmission of end-to-end control messages between H.323 endpoints. The H.245 protocol procedures establish channels for the transmission of audio, video, data, and control channel information. The transport of media in H.323 is provided by RTP and RTCP. As described in Section 2.3.1, RTP enables real-time end-to-end delivery of interactive audio, video and data.

2.3.3 Session Initiation Protocol (SIP)

The Session Initiation Protocol (SIP) is an application-layer signalling protocol defined by IETF RFC 2543 [29]. SIP is a signalling protocol for creating, modifying and terminating sessions, such as IP voice calls or multimedia conferences, with one or more participants in an IP network. The main advantages of SIP over other signalling protocols is that SIP offers a great deal of flexibility. The protocol is designed to be fast and simple. SIP provides: easy integration with existing IETF protocols, scalability and simplicity, mobility, and easy feature-and-service creation.

The SIP protocol provides the following functions [14]:

- *Name translation and user location*: SIP ensures that a call reaches the called party regardless of the party's location. SIP addresses are similar to e-mail addresses. Users are identified by a hierarchical URL based on elements such as a user's telephone number or host name.

- *Feature negotiation*: SIP allows all parties in a call to negotiate and agree on supported features. SIP recognizes that all participants in a call may not be able to support all features.

- *Call participant management*: During a session, a participant can bring other users into the call as well as transfer, hold or cancel any connections.

SIP Protocol Components

SIP consists of two basic components: the *user agent* and the *network server*. The user agent is the call endpoint, while the network server handles the call signalling associated with multiple calls. The user agent consists of the *user agent client (UAC)* which initiate calls, and the *user agent server (UAS)* which answers calls. The architecture of SIP allows peer-to-peer communication using a client/server protocol [14].

The network servers consist of four different types of servers:

- The *proxy server* acts as both a server and a client to make requests on behalf of other clients. Requests are processed by the proxy server internally, or by translating the request message and forwarding it to other servers.
- The *location server* is used to identify a called party's location.
- The *redirect server* accepts a SIP request, maps the address into zero or more new addresses, and returns these addresses to the client making the request. The redirect server does not initiate any requests of its own.
- The *register server* is a server that accepts SIP register requests, whereby a user indicates its availability at a particular address to the network. The register server is typically located with a proxy or redirect server.

2.3.4 MEGACO/H.248

While the H.323 protocol suite provides a gateway to interface with non-H.323 networks such as the PSTN, the protocol has been found to be not scalable within large public networks such as the Internet. Thus, a protocol was jointly developed by the IETF and ITU to address the issue of PSTN/VoIP integration. Known as MEGACO in the IETF and H.248 in the ITU-T, the MEGACO protocol evolved from the Media Gateway Control Protocol (MGCP), and is defined by RFC 3015 [30].

The MEGACO protocol removes the signalling control from the media gateway (MG) between two dissimilar networks, and places it in a media gateway controller (MGC). The

intelligence (control) is separated from the media (data). The MGC handles the control for one or more gateways. MEGACO is a master/slave protocol where the gateways simply execute instructions issued by the MGC. MEGACO/H.248 only specifies the communication protocol between the MGC and the MGs, and does not address communication between endpoints. The protocol interoperates with the peer-to-peer H.323 and SIP protocols. MEGACO provides a robust and flexible architecture and is easily scalable.

2.4 Chapter Summary

This chapter provided a general background of Voice over IP (VoIP) technology. Current trends indicate that the migration from the circuit-switched PSTN network to packet-based voice transmission is expected to continue in the foreseeable future. A generic VoIP transmission system was presented, from sampling the user's speech signal at one end to playing out the reconstructed voice stream at the other end. The main VoIP protocols reviewed were the TCP/UDP/IP protocols at the transport and network layers, respectively, and the RTP, H.323, SIP, and MEGACO/H.248 protocols at the application layer level.

Chapter 3

Delay Jitter Buffers

As the Internet is a best-effort delivery network, audio packets may be delayed and lost *en route* from sender to receiver. At the transmitter, speech/audio packets are generated at constant intervals and sent through the IP network to the receiver. The network delay experienced may vary for each packet depending on the paths taken by different packets and on the level of congestion at the routers along the path. The variation in network delay is referred to as *delay jitter*. Methods to reduce jitter can be divided into three types of approaches [36].

1. *Source-based approaches*: The transmitter will select the mode of the codec and adjust the bit rate according to current network conditions. The overall end-to-end delay, jitter and packet loss rate measure the level of congestion in the packet network [21].

2. *Network-based approaches*: Resources are reserved at network nodes to ensure that a certain Quality of Service (QoS) can be provided [37]. Similarly, real-time audio and video packets can be identified and given priority at nodes within the network [38].

3. *Receiver-based approaches*: Packets are typically buffered for a short period of time before they are played out. Late packets, i.e., packets that arrive at the receiver after their scheduled playout time, are considered 'lost'. By increasing the buffer delay, the number of late packets can be reduced, however the overall packet end-to-end delay increases. Hence, there is a tradeoff between the packet loss rate and the average packet end-to-end delay.

Source-based approaches require changes to be made in both the encoder used at the transmitter and in the decoder used at the receiver. Network-based approaches are costly as they require changes to be made to the existing network infrastructure; all nodes within the entire network will need to be updated. Receiver-based approaches are interesting, as they only require the addition of a playout buffer at the destination. Fig. 3.1 illustrates the role of the playout buffer in reducing jitter in a typical VoIP application.

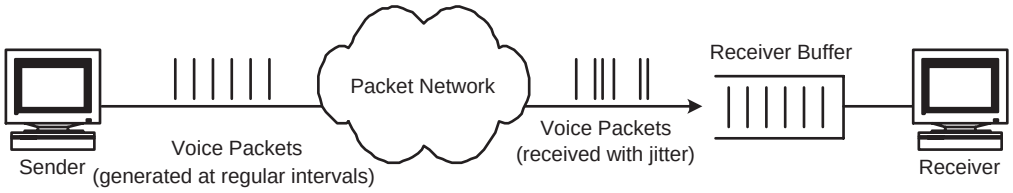


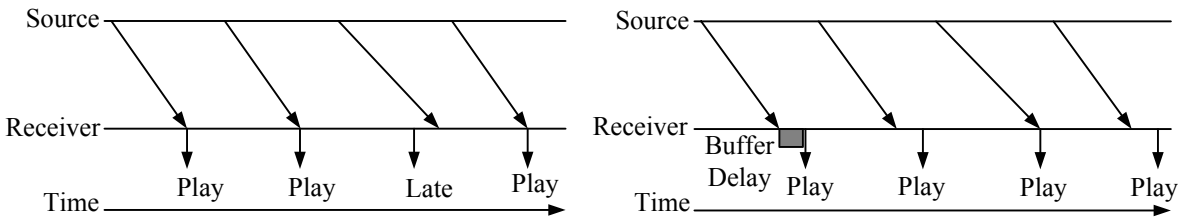
Fig. 3.1 Role of playout buffer in VoIP application [10]

Chapter 3 describes the challenge of jitter reduction through buffering. The role of playout buffer algorithms at the receiver is explained in Section 3.1. The main types of playout buffer algorithms are reviewed in Sections 3.1.1–3.1.4. A recent approach to adaptive playout, which performs the packet delay adjustment during talkspurts, is described in Section 3.2. Section 3.3 presents the proposed enhancement to the existing adaptive filter-based playout buffer algorithm.

3.1 Playout Buffer Algorithms

Fig. 3.2 illustrates the problem of jitter and underscores the need to employ playout buffer algorithms at the receiver. Audio packets are generated and sent through the network at periodic intervals. Due to the nature of the Internet, packets experience non-uniform network delays, and thus may arrive at the destination after they were supposed to be played out.

In the absence of a playout buffer, packets will be played out at the destination as soon as they are received. In Fig. 3.2(a), the arrival of the first packet will set the playout time. Subsequent packets will be played out at a periodic delay relative to the first packet. However, packets experiencing larger network delays will arrive too late to be played out. In this case, a large percentage of received packets may be lost and the quality of the audio played out at the receiver will be quite low. Buffering the packets for a short period of



(a) No buffering (b) Packets are buffered

Fig. 3.2 The playout buffering problem [12]

time as shown in Figure 3.2(b) ensures that a greater percentage of the received packets will arrive in time to be played out.

The total time between the generation and transmission of a packet at the sender (transmission time) and the packet being played out at the receiver (playout time) is referred to as the *end-to-end delay*. The end-to-end packet delay consists of *processing delay* to encode and packetize the audio signal, *network delay* incurred by the packet while traversing the network, and *buffering delay* experienced by the packet as it waits in the destination buffer before being played out.

Ideally, the end-to-end delay should be below 150 ms as delays of 150 ms cannot be perceived by humans [2]. End-to-end delays beyond 400 ms are irritating to users and impair interactivity in conversations [5]. Network delay measurements have been collected between various Internet hosts. Traces of network delay measurements are characterized by occurrences of large delay spikes. By decreasing the buffering delay, the end-to-end delay can be reduced at the cost of increased 'late' packet loss. The overall end-to-end delay will be tolerable to users and real-time Internet telephony can be achieved. Packets with end-to-end delays greater than the playout delay will still be 'lost', however loss rates of up to 20% may be tolerated when packet loss concealment (PLC) methods are applied [23]. Adaptive playout buffer algorithms attempt to adjust the playout delay for changing network conditions.

Fixed vs. Adaptive Playout Algorithms

Fixed playout algorithms estimate the minimum end-to-end delay that can be achieved for a specified loss rate [39]. This end-to-end delay is then fixed for the entire duration of the

voice call. However, network conditions fluctuate and by maintaining the fixed end-to-end delay, the packet loss rate may not remain at the specified value. Adaptive playout buffer algorithms attempt to adapt to changes in network conditions by dynamically adjusting the end-to-end delay, and hence are able to maintain a tolerable packet loss rate.

Playout Delay Adjustment Between Talkspurts

Since voice packets are generated at regular intervals, the received packets must be played out in a periodic manner. Playout delay adjustments made during periods of 'silence' (when the user is listening) between talkspurts (when the user is speaking), are less likely to be perceived by users. Therefore, the playout delay is adjusted on a per-talkspurt basis [39] by lengthening or shortening the periods of silence between talkspurts.

The basic playout buffer algorithm proposed by Ramjee *et al.* in [6] uses an autoregressive estimate to determine the playout for received audio packets. For the first packet in a talkspurt k, the playout time p_1^k is computed as

$$p_1^k = t_1^k + D^k, \tag{3.1}$$

where p_1^k and t_1^k are the playout time and sender timestamp, respectively, of the first packet in the k-th talkspurt, and D^k is the end-to-end delay for packets in the k-th talkspurt.

The end-to-end delay is fixed for subsequent packets in a talkspurt. Thus, the playout time for packet i in the k-th talkspurt is defined to be

$$p_i^k = t_i^k + D^k, \tag{3.2}$$

where t_i^k is the sending time of the i-th packet in the k-th talkspurt. It can also be seen that $D^k = p_1^k - t_1^k$. Therefore, the playout time for subsequent packets in a talkspurt can be calculated as an offset from the playout time of the first packet in the talkspurt,

$$p_i^k = p_1^k + (t_i^k - t_1^k). \tag{3.3}$$

Playout Delay Adjustment Within Talkspurts

In a recent approach to adaptive playout, playout delay adjustment is also performed within talkspurts. Individual voice packets are time-scaled such that they are played out just in

time for the predicted arrival time of the next packet. A dynamic time-scale modification technique [40] can be used to modify the playout rate while preserving the voice pitch. The degradation in perceptual quality due to scaling was found to be almost inaudible, even in extreme cases [40]. Dynamically adjusting the playout time improves overall performance by reducing end-to-end delay while keeping packet loss tolerable. Per-packet based playout delay adjustment will be described in Section 3.2. The main approaches to playout delay estimation are described here.

3.1.1 Autoregressive (AR) Estimate-Based Algorithms

The basic playout algorithm proposed as Algorithm 1 in [6] uses an autoregressive approach to estimate the network delay and jitter. The algorithm to estimate network delays is based on the method described in the specification for the transmission control protocol (TCP) in RFC 793 [17]. Likewise, the algorithm used to determine network jitter is based on Jacobson's method [41] to measure the variation in network delays.

The estimate for the average network delay, $\hat{d}_i$, is

$$\hat{d}_i = \alpha \hat{d}_{i-1} + (1 - \alpha) n_i, \tag{3.4}$$

where $\hat{d}_i$ is the autoregressive estimate of the packet delay, n_i is the network delay incurred by the i-th packet, and α is a weighting factor used to control the convergence rate of the algorithm.

The variation in the network delay, $\hat{v}_i$, is similarly estimated,

$$\hat{v}_i = \alpha \hat{v}_{i-1} + (1 - \alpha) |\hat{d}_i - n_i|. \tag{3.5}$$

The total end-to-end delay, D_i is then calculated as

$$D_i = \hat{d}_i + \beta \hat{v}_i, \tag{3.6}$$

where β is a safety factor used to moderate the tradeoff between end-to-end delay and packet 'loss' rate due to late packets. The $\beta \hat{v}_i$ term is a safety buffer term used to ensure that the end-to-end delay is large enough, so that only a small portion of received packets will arrive too late to be played out. A higher value of β results in a lower loss rate as more

packets arrive in time, however the total end-to-end delay increases.

The values of α and β are set to 0.998002 and 4.0, respectively in Algorithm 1. While the AR estimates of the delay, $\hat{d}_i$, and the variation, $\hat{v}_i$, are updated for each packet, the total end-to-end delay, D^k, is only set to D_i at the beginning of a new talkspurt.

Modifications to Algorithm 1

The second algorithm proposed in [6] is a slight modification to the first algorithm. Based on a suggestion by Mills in RFC 889 [42], two different values are used for α, one for increasing network delays and the other for decreasing delays. The algorithm attempts to rapidly adapt to increases in network delay by using a value of 0.75 for α when the network delay, n_i, is greater than the AR delay estimate, $\hat{d}_i$. When the network delay is decreasing, i.e., $n_i < \hat{d}_i$, α is kept at 0.998002 as in Algorithm 1. The network delay estimate, delay variation estimate, and end-to-end delay are computed using the appropriate value for α in Eqs. (3.4), (3.5), and (3.6) from Algorithm 1.

The third algorithm proposed in [6] simply sets the delay estimate, $\hat{d}_i$, to the minimum of all network delays in the previous talkspurt,

$$\hat{d}_i = \min_{j \epsilon S_i} \{n_j\}, \tag{3.7}$$

where S_i is the set of all packets received in the talkspurt prior to the one initiated by packet i. The variation in network delay, $\hat{v}_i$, and end-to-end delay, D_i, are once again computed using Eqs. (3.5) and (3.6).

α-Adaptive Algorithm

The choice of the value of α in Algorithm 1 determines the extent of the effect of the recent network delay value, n_i, on the AR estimate of the delay, $\hat{d}_i$. Minor changes in the value of α can greatly impact the tradeoff between packet loss and total end-to-end delay. An improvement can be expected if the value of α itself is adaptively adjusted to suit the changing network conditions [43].

Algorithm 1 is modified in [44] to adaptively adjust α to an optimal value. The algorithm calculates the loss rate for previous talkspurts based on the current value of α. It then either increments or decrements the value of α. The playout delay is adjusted for each talkspurt.

The increment value is much smaller than α as small changes in α lead to large changes in packet loss and total end-to-end delay. The α-adaptive playout algorithm performs better than both Algorithms 1 and 2, showing large reductions in end-to-end delay in regions of low loss [44].

3.1.2 Statistically-Based Algorithms

Statistically-based approaches use the statistics of past network delays to compute the current packet playout delay. Based on the distribution of past delays, the playout delay is selected such that a tolerable percentage of packets will arrive 'late'.

Adaptive Gap-Based Algorithm

The adaptive gap-based algorithm [10] stores the network delay values for a talkspurt and computes the optimum playout delay for the talkspurt. For each talkspurt, the optimum theoretical playout delay is defined to be the minimum playout buffering delay resulting in the specified packet loss rate for that talkspurt. This theoretical quantity cannot be calculated until the talkspurt is finished. The playout delay for the next talkspurt is then increased or decreased to the minimum playout buffering delay that was calculated for the previous talkspurt.

Histogram Approach

Algorithm 3 from [9] logs the delay of each packet and updates a histogram of packet delays after every packet arrival. The algorithm computes the playout delay, D^k, for the new talkspurt by finding the delay `curr_delay` for a given percentile point q in the distribution function [9]. The distribution function uses the packet delays for the last w packets to form a histogram. The pseudocode is shown here.

```
// Algorithm 3 from [9]
IF (delays[curr_pos] ≤ curr_delay)
   count -= 1;
distr_fcn[delays[curr_pos]] -= 1;
delays[curr_pos] = n_i;
curr_pos = (curr_pos+1) % w;
```

```
distr_fcn[n_i] +=1;
IF (delays[curr_pos] < curr_delay)
   count += 1;
WHILE (count < w × q)
   curr_delay += unit;
   count += distr_fcn[curr_pos];
WHILE (count > w × q)
   curr_delay -= unit;
   count -= distr_fcn[curr_pos];
D^k = curr_delay;
```

The number of packet delays, w, stored in the histogram determines how sensitive the algorithm is in adapting to changing network conditions. If w is too small, the algorithm will have a myopic view of the past, and will likely produce a poor estimate of the playout delay [9]. On the other hand, if w is too large, the algorithm will keep track of a large amount of history, and will not be able to react quickly to changes. It was found that for $w < 10,000$, the performance degraded as the number of packets stored in the histogram decreased. For values above 10,000 packets, any performance enhancement was marginal, thus in [9], w was set to 10,000 packets.

Probable Packet Delay Distribution

Another statistically-based method constructs a *packet delay distribution* (PDD) curve. The PDD is an estimate of the probable delays suffered by packets in the network over a time window. It may draw on existing traffic conditions, history information, or any negotiated service characteristics to derive initial estimates for delay bounds and distributions [11]. The PDD constructed from this information is approximate, but over time it is updated dynamically so that it closely tracks network performance. For a given PDD, packets with network delays less than the playout delay will be played out, and those with network delays greater than the playout delay will be declared late.

To approximate the PDD curve, the approach advocated in [11] is to store and track network delay trends using a measured histogram. Since network characteristics change over time, statistical trends vary and current information is necessary for the prediction to

be accurate. Hence, each bin in the histogram needs to be *aged* or updated to diminish the effect of older samples.

Aging Past Packet Delays

There are two basic approaches to aging [11].

- *Full Aggregation* method: Data is accumulated into a probability distribution curve throughout the lifetime of the transmission. Both recent and old information are weighted equally.
- *Flush and Refresh* method: Data is stored for a period of time, and then periodically flushed and refreshed. The flush results in a complete loss of historic information.

The aging function used in [11] gradually ages older samples. Instead of discarding older information, the information is gradually retired.

The aging approach used gradually diminishes the effect of older samples on the distribution by periodically scaling down the existing distribution by an aging factor [11]. Three variations are presented in [45]. In Algorithm 1, each bin count is multiplied by a constant aging factor and the bin containing the delay of the current packet is then incremented by one. As the number of data samples increase, the corresponding bin counts increase, thus newly arrived packets contribute less to the histogram in comparison to older data. For example, suppose the total value of all bin counts in a histogram is 10, it is scaled down by 0.9 after aging. The new packet delay then contributes 1 to the histogram, thus the ratio of old data to the new packet delay is 9:1. However, after some time, the value of all bins in the histogram may be 10,000. After aging by 0.9, the ratio of old data to new is now 9,000:1, thus reducing the impact of the new packet delay. In Algorithm 2, the ratio of old aged data to the newly arrived packet delay is kept constant through the lifetime of the stream. Thus, the aging factor is a function of the total histogram count. Algorithm 3 extends this further by considering the frequency at which aging is performed. Even if the aging frequency is changed dynamically, the ratios are kept constant between each occurrence of aging.

Cumulative Distribution Function

An adaptive packet-based adjustment scheme is proposed in [7]. When packet i arrives, its network delay, d_i, is used together with past delays to estimate the delay of the next packet, $\hat{d}_{i+1}$. The current packet i is then time-scaled so that it finishes playing out just as packet $(i+1)$ is expected to arrive. The time-scale modification technique is described in Section 3.2.2. To estimate the current packet delay, the statistics of past delays are used. Using a sliding window of the past w packets, the past delays are collected and sorted in ascending order from d_1 to d_w, so that a cumulative distribution function (cdf) can be constructed. Estimates of the lowest possible packet delay d_0 and the maximum delay d_{w+1} are also computed using:

$$\begin{aligned} d_0 &= \max(d_1 - 2s_{d_w}, 0), \\ d_{w+1} &= d_w + 2s_{d_w}, \end{aligned} \tag{3.8}$$

where s_{d_w} is the standard deviation of the past w packet delays. Due to the heavy-tailed nature of network delay, the maximum delay value cannot be determined from a limited sample space. Hence, the statistics obtained from the last w samples are often too optimistic. By adding an estimate of the maximum delay, d_{w+1}, the sample becomes more conservative. The estimate of the next packet delay can then be obtained from the cdf for the desired loss rate.

Modelling Packet Delay Distribution with a Pareto Distribution

Another statistically-based approach [46] analyzes the tail part (95–99.9% region) of a packet delay distribution and finds that it is best modelled by a Pareto distribution. The Pareto distribution is given by

$$F(x) = 1 - \left(\frac{k}{x}\right)^{\alpha}, \quad x \geq k. \tag{3.9}$$

where α and k are distribution parameters [46].

On each packet arrival, the parameters of the Pareto cumulative density function, $F(x)$, are updated. The playout delay, $\hat{d}_i$, is then estimated from the equation $F(\hat{d}_i) = X$, where X is the target point in the distribution [47]. The proposed algorithm gives the playout

delay, $\hat{d}_i$, to be

$$\hat{d}_i = k\left(1 - \frac{X}{100}\right)^{\alpha}. \tag{3.10}$$

By utilizing the mean opinion score (MOS) function, a subjective evaluation of audio quality, the Pareto distribution-based playout algorithm [47] is enhanced in [48] by considering the user's perceived quality for real-time VoIP applications. Rather than minimizing the playout delay for a desired loss rate, the algorithm selects the playout delay and corresponding late packet percentage, which maximize the user-perceived quality or MOS value.

3.1.3 Adaptive Filter-Based Algorithms

A novel approach to adaptive playout buffer algorithms is proposed in [12]. Instead of reacting to network fluctuations, the approach is to predict the network delay. The playout delay is then determined from the predicted network delay and variation. An accurate prediction of the network delay can rapidly track network changes and thus adjust the delay more effectively.

Adaptive filtering algorithms have traditionally been used for equalization and echo cancellation. The basic adaptive filtering algorithm aims to minimize the expected mean square error between the actual data and the estimate [13]. Previous data are passed through a finite-impulse response (FIR) filter to compute the current estimate. The mean square error is then used to adjust the tap weights of the adaptive filter. The block diagram of an adaptive filter is illustrated in Figure 3.3.

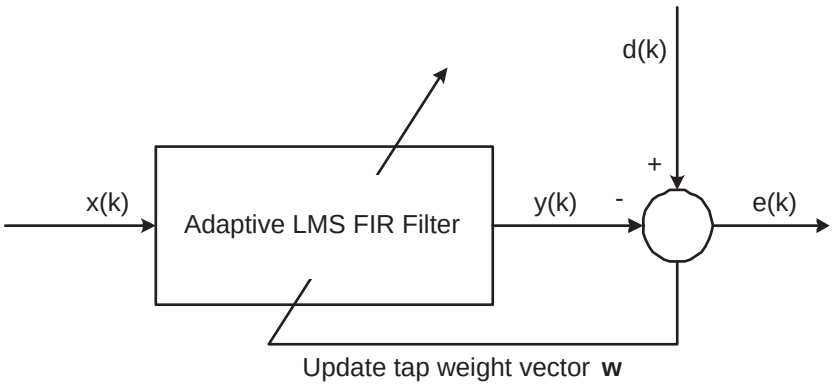


Fig. 3.3 Block diagram of adaptive LMS filter

NLMS Algorithm

The normalized least mean square (NLMS) algorithm is used for the adaptive predictor in [12]. The estimate for the network delay, $\hat{d}_i$, is computed to be

$$\hat{d}_i = \mathbf{w}_i^T \mathbf{n}_i, \tag{3.11}$$

where $\hat{d}_i$ is the predicted network delay value for the i-th packet, $\mathbf{w}_i$ is the $N \times 1$ vector of adaptive filter coefficients, $()^T$ is the vector transpose, and $\mathbf{n}_i$ is the $N \times 1$ vector containing the past N network delays (up to and including the delay for packet $(i-1)$).

The filter tap weights, $\mathbf{w}_i$, are then updated after each packet using the NLMS algorithm [13].

$$\mathbf{w}_{i+1} = \mathbf{w}_i + \frac{\mu}{\mathbf{n}_i^T \mathbf{n}_i + a} \mathbf{n}_i e_i, \tag{3.12}$$

where μ is the step size, a is a small constant to prevent division by zero, and e_i is the estimation error.

The estimation error, e_i, is given by

$$e_i = \hat{d}_i - n_i, \tag{3.13}$$

where n_i is the actual network delay, and $\hat{d}_i$ is the predicted delay for packet i. Table 3.1 lists the values of the parameters used in [12].

Table 3.1 NLMS algorithm parameter values used in [12]

Parameter	*Value*
$\mathbf{w}_0$	$[1\ 0\ \ldots\ 0]^T$
N	18
μ	0.01

The network delay variation, $\hat{v}_i$, and total end-to-end delay, D_i, are calculated using Eqs. (3.5) and (3.6) from the basic autoregressive algorithm [6].

A set of packet traces was collected and analyzed using both the reactive algorithm proposed in [6] and the NLMS algorithm. The traces did not distinguish between talkspurts and silence periods. For both the reactive and the NLMS algorithms, the total end-to-end delay, D_i is computed on a per-packet basis. The results indicate that the NLMS

predictor [12] reduces the total end-to-end delay for the sample traces while maintaining very low packet loss rates.

Addition of Decorrelation Filter to NLMS Algorithm

An enhancement to the NLMS algorithm is proposed in [44]. The performance of the NLMS filter can be improved if the data passed though it is decorrelated, resulting in lower errors and faster convergence of the NLMS algorithm. The Discrete Wavelet Transform (DWT) is suggested for decorrelation. The Beylkin, Daubechies and Haar wavelets were all experimented with, and as the choice of wavelet does not significantly affect the decorrelation, the simple Haar wavelet transform is used. The order of the decorrelating filter depends on how far into the past the correlation extends. The correlation $r_N(k)$ can be estimated from the covariance lags, as follows.

$$r_N(k) = \frac{1}{N} \sum_{j=0}^{N-k-1} n_{j+k} n_j, \quad k = 0, 1, \ldots, N/2, \tag{3.14}$$

where n_i is the network delay for packet i, and N is the number of delay samples used.

The NLMS algorithm is implemented in [44] on traces from [6] on a per-talkspurt basis rather than on a per-packet basis. Simulations show that the parameter values for μ and N used in [12] did not necessarily work well on traces obtained elsewhere. The values of the parameters were adjusted, and it was found that the performance of the NLMS predictor is no better than that of the autoregressive-based algorithm for talkspurt-based adjustment. The addition of the decorrelation filter improved the performance of NLMS. As the entire trace is not available in practice, it is suggested to estimate the statistics of the network delay trace from the first few packets and to use these to set the values of various NLMS algorithm parameters [44].

3.1.4 Delay Spikes

Various traces of network delays and loss have been collected while developing playout buffer algorithms. Some of the traces are characterized with occurrences of network delay spikes. The frequent occurrences of spikes have been reported by [42, 49]. A spike begins with the sudden onset of a large increase in network delay. Although subsequent packets usually experience declining network delays, their delay values are still quite large. The

spike ends when network delays return to a steady-state value. Delay spikes can be caused by heavy congestion resulting in long queues at routers within the network. Typical delay spikes are depicted in Fig. 3.4.

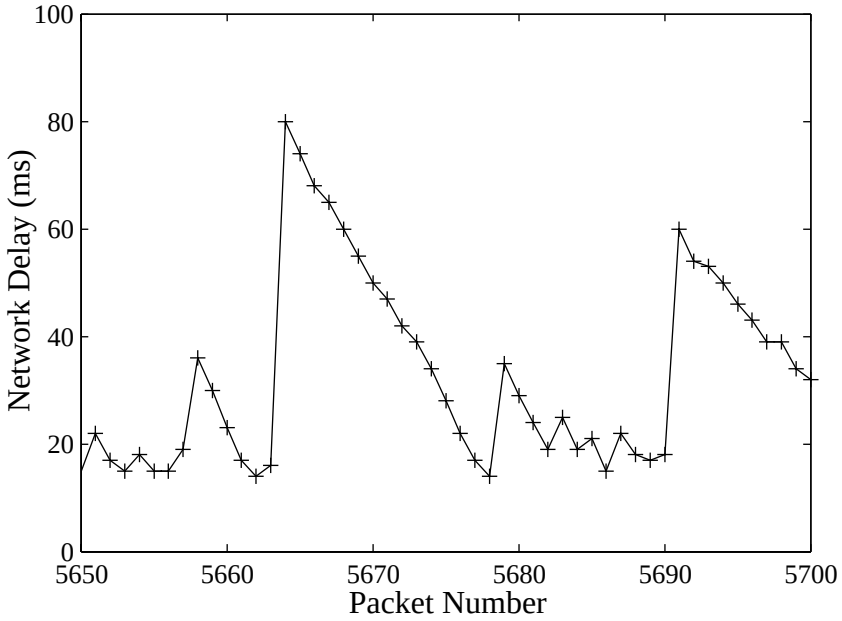


Fig. 3.4 Typical network delay spike

Spike Detection Mode

The first three AR-based playout algorithms described in [6] are unable to adapt rapidly to delay spikes. The autoregressive estimate-based approaches do not increase their delay estimates quickly enough in the presence of spikes, and take too long to reduce their delay estimates after the completion of a spike. A fourth algorithm was thus developed in [6] to adapt to such spikes and is described here.

Algorithm 4 is the first method to introduce the notion of a spike or impulse mode [6]. Upon detecting a delay spike, the playout algorithm switches to an impulse or spike mode. Within the spike, the delay estimate is only dependent on the most recent network delay values. When the spike is completed, the algorithm reverts to normal mode and the delay estimate is computed using the autoregressive estimate-based approach from Algorithm 1.

A delay spike is characterized by a sudden large increase in delay. Therefore, a delay spike can be detected when the difference in delay values for the two most recently received

packets is greater than a certain threshold. It is a bit more difficult to determine the end of a spike. In [6], the slope of the delay spike is monitored. As the spike completes and the delays flatten out, the slope will reduce and fall below a threshold. The playout algorithm then reverts back to normal mode. The pseudocode of Algorithm 4 is given here.

```
// Algorithm 4 from [6]

1. n_i = Receiver_timestamp - Sender_timestamp;

2. if(mode == NORMAL){
      if(|n_i - n_{i-1}| > 2|v̂| + 100) {
         var = 0; /* Detected beginning of spike */
         mode = IMPULSE;
      }
   else {
      var = var/2 + |2n_i - n_{i-1} - n_{i-2}|/8;
      if(var <= 8) {
         mode = NORMAL; /* End of spike */
         n_{i-2} = n_{i-1};
         n_{i-1} = n_i;
         return;
      }
   }

3. if(mode == NORMAL)
      d̂_i = 0.125n_i + 0.875d̂_{i-1};
   else
      d̂_i = d̂_{i-1} + (n_i - n_{i-1});
   v̂_i = 0.125|n_i - d̂_i| + 0.875v̂_{i-1};

4. n_{i-2} = n_{i-1};
   n_{i-1} = n_i;
   return;
```

Adaptive Playout with Spike Detection

Many of the playout buffer algorithms described in the previous sections have also incorporated spike detection and compute their delay estimates differently during spikes. The adaptive gap-based algorithm [10] simply incorporates the spike detection mode from [6]. Algorithm 3 from [9] also works in two modes. In normal mode of operation, the distribution of packet delays is updated and the algorithm computes the delay estimate using the histogram-based approach. However, if a packet arrives with a delay that is larger than some multiple of the current playout delay, the algorithm switches to spike mode. Within the spike, packet delays are no longer collected and the algorithm conservatively sets the playout delay to the first packet delay recorded in the spike. The end of the spike is detected when the delay of a newly arrived packet is less than some multiple of the playout delay before the current spike. The mode is then set back to normal.

The per-packet adjustment playout algorithm also implements a spike or *rapid adaptation* mode [7]. The playout algorithm switches to rapid adaptation mode when the present delay exceeds the previous one by more than a threshold value. In rapid adaptation mode, the first packet of the spike has to be discarded. The delay estimates for subsequent packets in the spike are set to the high delay value of the first packet in the spike. The delay statistics are not updated within the spike. The algorithm returns to normal mode of operation when packet delays fall back to the level before spike adaptation mode, and the previously stored delay statistics are reused.

3.2 Playout Delay Adjustment Within Talkspurts

Since voice packets are generated at periodic intervals at the transmitter, the receiver must play the packets out continuously at the same rate. The received packets are buffered and scheduled for playout at regular intervals in the future. Conventional adaptive playout algorithms adjust the playout delay by lengthening or shortening the periods of silence between talkspurts. Playout delay adjustments made during periods of silence are unlikely to be perceived by users.

In contrast to previous work, a new playout scheduling algorithm [40] adjusts the playout delay not just between talkspurts, but also within talkspurts. The proper reconstruction of continuous speech is achieved by scaling individual voice packets using a time-scale

modification technique based on the WSOLA algorithm [8], which adjusts packet size while preserving voice pitch. Subjective listening tests have shown that the dynamic time-scale modification of voice packets does not impair audio quality [40].

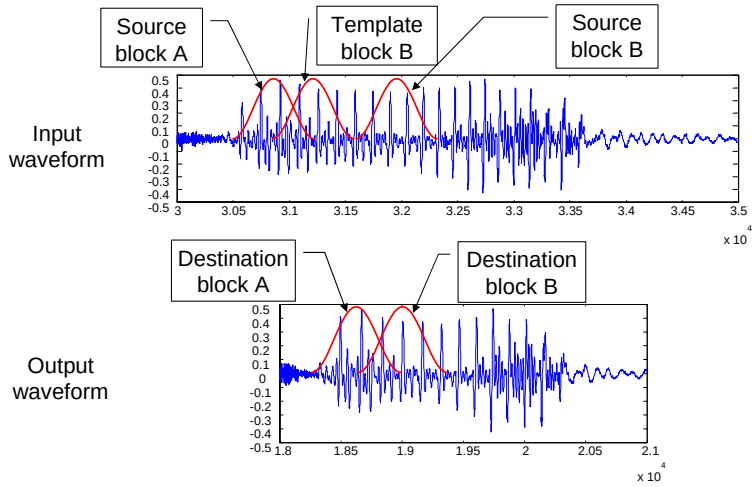
Section 3.2.1 describes the Waveform Similarity Overlap-Add (WSOLA) algorithm and Section 3.2.2 describes the modifications made in [7, 40] to the WSOLA algorithm to make it suitable for scaling audio packets. The modified algorithm is referred to as the packet-based WSOLA algorithm.

3.2.1 WSOLA Algorithm

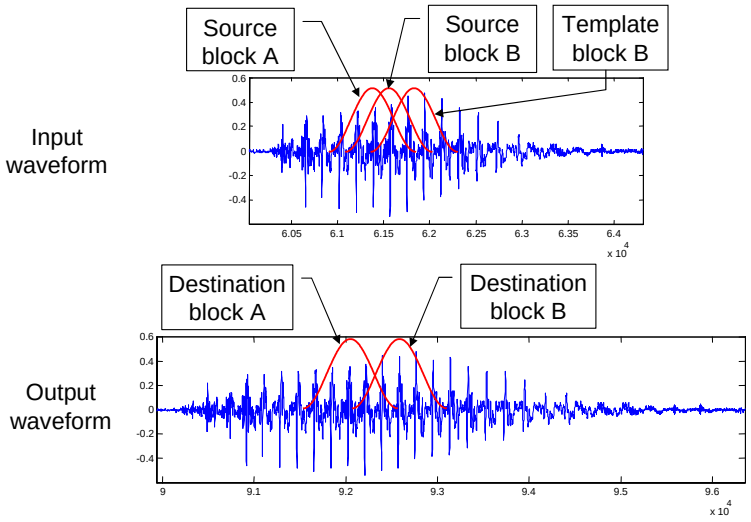
The Waveform Similarity Overlap-Add (WSOLA) algorithm proposed in [8] is a robust and computationally efficient algorithm used for high quality time-scale modification of speech. Time-scale modification techniques aim to change only the apparent speaking rate, while preserving other perceived aspects of speech such as timbre, voice quality, and pitch. The basic idea of WSOLA is to decompose the input into overlapping segments of equal length, which are then realigned and superimposed with fixed overlap to form the output. The realignment leads to an increase or decrease in the output length. Specifically, WSOLA produces a synthetic waveform, $y(n)$, that maintains maximal local similarity to the original waveform, $x(m)$, in the neighbourhoods of all sample indices given by the mapping, $n = \tau(m)$, where $\tau(m)$ is the transformation function defined as $\tau(t) = \alpha t$, α being the time-scaling factor. If $\alpha > 1$, the output speech is stretched, and if $\alpha < 1$, the output speech is compressed.

The WSOLA algorithm operates entirely in the time domain. The algorithm works by segmenting the input audio waveform into blocks of equal length. Audio blocks in the input waveform are selected and overlap-added to produce the output audio. If the source blocks were taken at regular intervals in the original waveform, the output file would be of poor quality as the pitch pulses are not equally spaced. Thus, the selection of similar source blocks in the input to use for overlap-add, is critical to achieving high output quality.

Fig. 3.5 illustrates the basic operation of the WSOLA algorithm. The algorithm iteratively constructs the output waveform, block by block. In Fig. 3.5, source block A is copied to the destination block A. Template block B is the block following source block A with 50% overlap. WSOLA now needs to find a block to copy to destination block B to overlap-add with destination block A. Therefore, source block B is desired to closely



(a) WSOLA compression at $\alpha = 0.6$



(b) WSOLA expansion at $\alpha = 1.5$

Fig. 3.5 Illustration of WSOLA algorithm

resemble template block B. The reverse transformation, $\tau^{-1}(t) = \frac{1}{\alpha}t$, gives the centre of the search region in which to look for source block B. A measure of waveform similarity is computed between template block B and blocks in the search region. The source block with the greatest similarity is then copied to destination block B. The template block for the next iteration will be the block right after source block B with 50% overlap. For a given iteration, the source block follows the template block in WSOLA compression, and it precedes the template block in WSOLA expansion, as shown in Figs. 3.5(a) and 3.5(b), respectively.

Once the positions of the template block and the search region are known, a series of correlations is computed between the template block and blocks in the search region. Each source block in the search region is shifted by δ, where $-L/2 \leq \delta \leq L/2 - 1$, and L is the length of the search region.

The similarity measure used in this work is the cross-correlation coefficient,

$$\rho(\delta) = \sum_{k=0}^{N} TemplateBlock(k) \times SourceBlock(k + \delta), \tag{3.15}$$

where N is the length of a block. The weighting window used in this work for the overlap-add operation is the full raised-cosine window, $h(n) = 0.5 - 0.5\cos(\frac{2\pi n}{N})$, with 50% overlap. The window size is set to be the length of a block, N.

The speech quality and algorithm computation time are affected by the block size and the length of the search region for the source block. Larger blocks contain more pitch periods so the correlations will give a better measure of the waveform similarity between template and source blocks. However, if the block size is too large, artifacts such as echoes and tinny sounds will be introduced into the output. A larger search region results in more correlations being computed, thus it is computationally more expensive. Nevertheless, a better match with higher correlation between template and source block may be found within a larger search region.

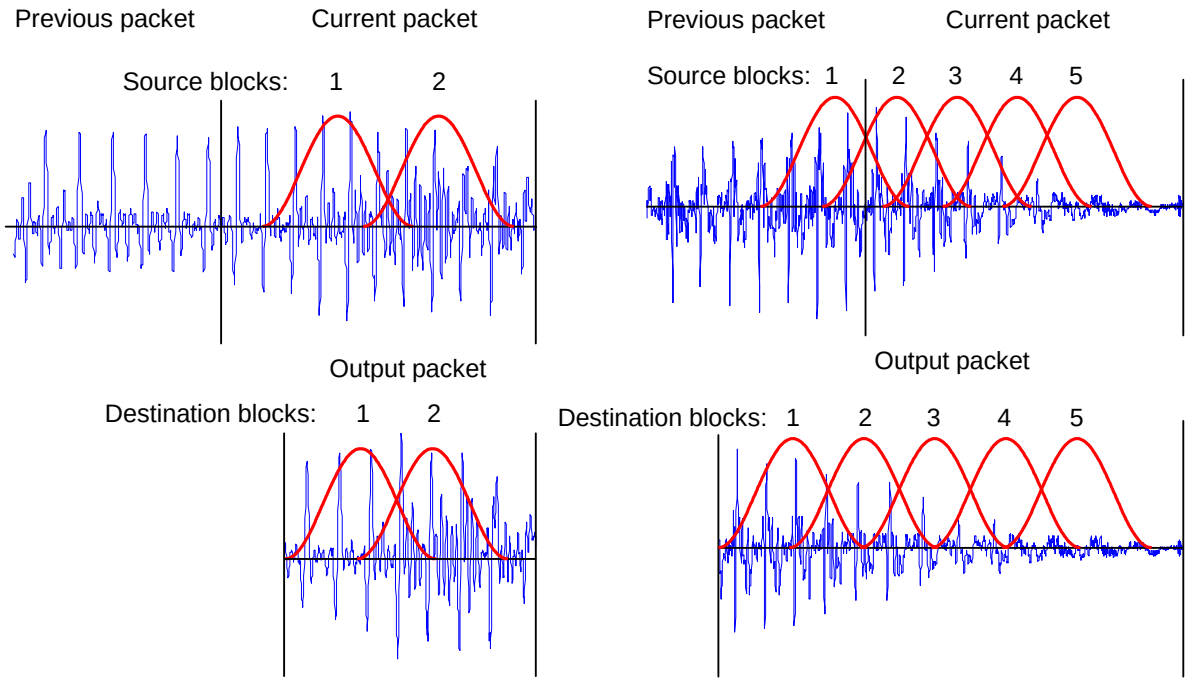
3.2.2 Packet-Based WSOLA

The WSOLA algorithm was tailored by Liang in [7, 40] to work on a single packet. The main modifications made by the packet-based WSOLA algorithm are reducing the block length for correlation calculations and including the previous packet in the search region

of the source block to be overlap-added with the first template segment of the packet.

The search for a source block is constrained when working on a single packet as the realignment of segments must be done in units of pitch periods. As there are few pitch periods in a short packet, it becomes difficult to find a similar waveform segment in the input using correlation calculations. Thus, the length of the correlation calculations was reduced to use only **half** the block length instead of the whole block length.

In the packet-based WSOLA algorithm, the first template segment is positioned at the beginning of the input packet. In order to have a larger range to search for similar waveforms in packet expansion, the search region for the first source block is chosen inside the previous packet. Although the previous packet has already been played out at the time of scaling, similar waveforms can still be found within the packet, and they can be used to construct the current packet output. If the packet is to be compressed, the first source block is searched inside the packet. Figs. 3.6(a) and 3.6(b) illustrate packet-based WSOLA compression and expansion, respectively.



(a) Packet-based WSOLA compression

(b) Packet-based WSOLA expansion

Fig. 3.6 Packet-based WSOLA algorithm

To ensure continuity at packet edges, the last samples in an output packet are kept the same as those in the input. As shown in Fig. 3.6, audio samples after the last source block in the input are copied directly to the output packet. A scaled packet can thus be played out at the receiver without needing to smooth the transition to the next packet.

The packet-based WSOLA time-scale modification technique is implemented in this work. Informal listening tests conclude that users cannot discern any difference in audio quality. Subjective results presented in [40] also show that the degradation in audio quality was found to be almost inaudible, even in cases with excessive time-scaling of packets. The packet-based WSOLA algorithm is thus well-suited for playout delay adjustment on a per-packet basis. The dynamic time-scale modification of individual voice packets allows adaptive playout algorithms to rapidly adjust the playout delay to changing network conditions.

3.3 Adaptive NLMS Playout Algorithm with Delay Spike Detection

The autoregressive estimate-based algorithms measure the network delay and variation and *react* to changes in network conditions. Similarly, statistically-based algorithms adjust the playout delay based on *past* network packet delays. The newer packet-based playout delay adjustment schemes allow increased flexibility in playout scheduling algorithms. Playout buffer algorithms which attempt to *predict* the network delay and variation can take advantage of this flexibility. These algorithms can time-scale individual voice packets such that the i-th packet finishes playing out just as the $(i+1)$-th packet arrives. An accurate prediction of the network delay will allow the playout delay to be adjusted effectively by closely tracking fluctuations in network delay and thus reducing the 'loss' due to late packet arrivals.

The adaptive filter-based NLMS algorithm aims to make an accurate prediction of the network delay, and is thus well-suited for newer playout algorithms where playout delay is adjusted per individual packet. The adaptive playout algorithm proposed in this work is based on the NLMS predictor described previously in Section 3.1.3. The basic NLMS predictor is improved here by introducing a spike-detection mode to rapidly adjust to delay spikes.

3.3.1 Existing NLMS Predictor

The NLMS adaptive filtering algorithm [12] provides a good *prediction* of the network delay and closely tracks any fluctuations in network delay. However, a drawback of the NLMS predictor is that it does not explicitly detect delay spikes and therefore does not alter its behaviour during a spike.

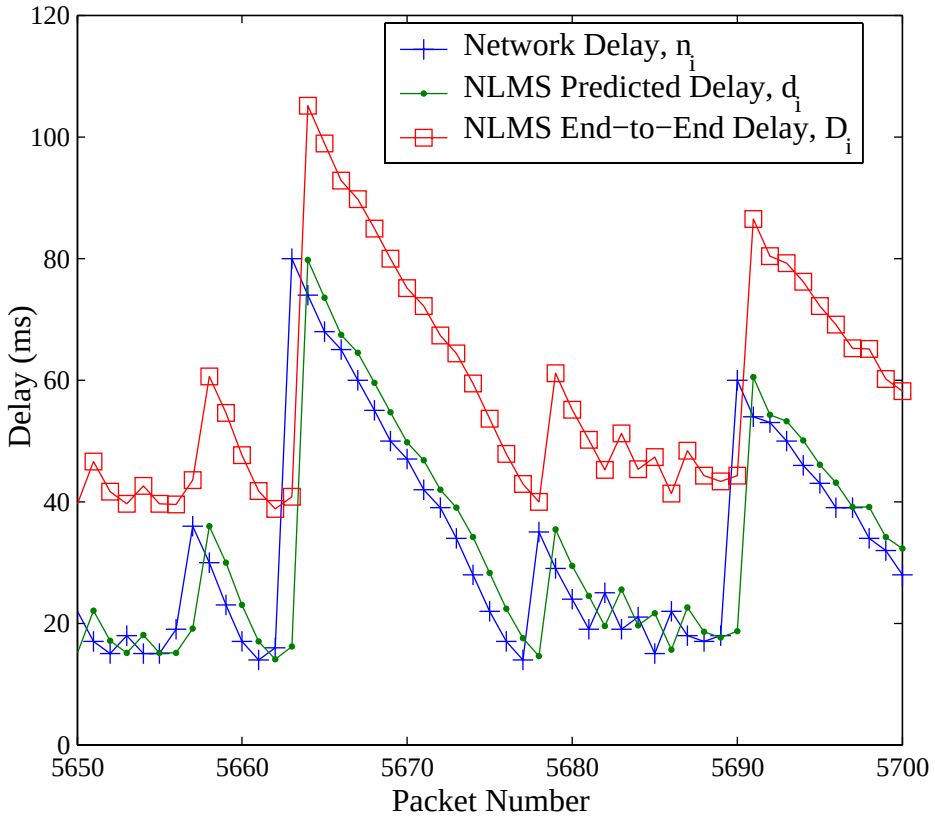


Fig. 3.7 NLMS playout algorithm

Fig. 3.7 illustrates the behaviour of the NLMS algorithm in the presence of a delay spike. The end-to-end delay, D_i, is the predicted network delay, $\hat{d}_i$, plus the safety buffer term $\beta\hat{v}_i$, as given by Eq. 3.6. It is difficult to predict when a delay spike will occur [42]. The first packet in a delay spike will arrive sometime after it has been scheduled to be played out, and thus will be considered late or 'lost'. The NLMS predictor will react to the large increase in network delay, and subsequent delay predictions will overestimate the ensuing packet delays. The overshoot in the end-to-end delay can be reduced by decreasing the value of the safety factor, β, thereby reducing both the safety buffer term, $\beta\hat{v}_i$, and the

end-to-end delay, D_i.

3.3.2 Enhanced NLMS (E-NLMS) Algorithm

An improved NLMS algorithm called the enhanced NLMS or E-NLMS algorithm is proposed in this work [50]. The E-NLMS algorithm improves the basic NLMS predictor by adding a spike-detection mode to the algorithm.

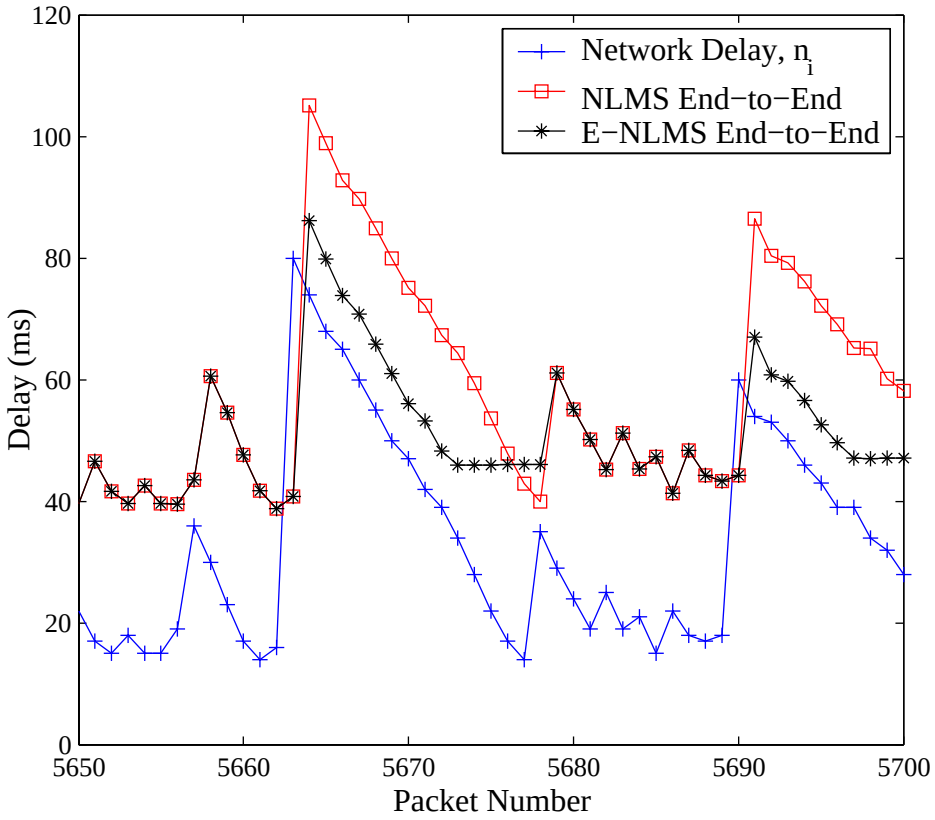


Fig. 3.8 NLMS and E-NLMS playout algorithms

During normal mode of operation, the E-NLMS algorithm works exactly like the basic NLMS predictor [12]. A delay spike is detected when either the previous packet was late (it arrived after it was supposed to be played out) or the actual network delay exceeds the predicted delay value by a threshold. Upon switching to spike mode, the playout delay is still based on the NLMS delay prediction. However, as the NLMS algorithm overestimates the network delay for packets following the beginning of a delay spike, the value of the safety factor, β, in Eq. (3.6) can be decreased, thereby reducing both the safety buffer

term, $\beta\hat{v}_i$, and the end-to-end delay, D_i, during the spike. The end of the spike is detected and the mode of operation switches back to normal mode when the NLMS delay prediction, $\hat{d}_i$, no longer exceeds the actual network delay, n_i.

Fig. 3.8 illustrates the behaviour of the enhanced bi-modal NLMS algorithm. As a spike is characterized by a sudden, large increase in network delay followed by declining network delays, the bi-modal algorithm takes advantage of this trait by reducing the safety buffer term, and thus tracks the network delay more accurately. To avoid having the playout delay fall too low, the end-to-end delay, D_i, is not allowed to fall below the delay estimate computed using Eqs. (3.4), (3.5) and (3.6) from the basic autoregressive algorithm [6]. The pseudocode of the algorithm is given here.

```
// Enhanced NLMS (E-NLMS) algorithm
// Estimate Delay using NLMS Filter
d̂_i = w_i^T n_i;

ARdelay_i = αARdelay_{i-1} + (1 − α)n_{i-1};
v̂_i = αv̂_{i-1} + (1 − α)|d̂_{i-1} − n_{i-1}|;

if ( mode == SPIKE )
    varfactor_i = β/4  v_i;
    D_i = d̂_i + varfactor_i;
    if (D_i < ARdelay_i + βv̂_i)
        D_i = ARdelay_i + βv̂_i;
    end
else // Normal mode
    varfactor_i = βv̂_i;
    D_i = d̂_i + varfactor_i;
end

// if end-to-end delay < network delay
if (D_i < n_i)
    packet_i=LOST;
else
```

$packet_i$=*IN_TIME*;
end

if $(n_i > \hat{d}_i)$
    mode = *NORMAL*;
end
if($(n_i > \hat{d}_i + 5\hat{v}_i)$ OR $(packet_i == LOST)$)
    mode = *SPIKE*;
end

// *Update Adaptive NLMS Filter Tap Weights*
$e_i = \hat{d}_i - n_i$
$\mathbf{w}_{i+1} = \mathbf{w}_i + \mu/(\mathbf{n}_i^T \mathbf{n}_i + a)\mathbf{n}_i e_i$;

3.4 Chapter Summary

This chapter introduced the various approaches to reducing *jitter* or variation in network delay. The role of the playout buffer in receiver-based approaches was discussed and the main types of playout buffer algorithms were reviewed. The main playout buffer algorithms are not robust enough to adapt their network delay estimates in the presence of delay spikes, thus spike detection has been incorporated in some playout buffer algorithms.

Traditionally, playout buffer algorithms have adjusted the playout delay during periods of silence by expanding or compressing the length of silence between talkspurts. Newer playout algorithms adjust the playout delay on an individual packet basis by dynamically time-scaling individual voice packets. Packet-based adjustment algorithms are more flexible as they can take advantage of accurate network delay estimates.

The adaptive NLMS predictor uses an accurate prediction of the network delay to closely track network fluctuations. The proposed algorithm, the enhanced NLMS (E-NLMS) algorithm improves the basic NLMS predictor by introducing a spike detection mode to rapidly adjust the playout delay during delay spikes. In Chapter 4, the E-NLMS algorithm is evaluated in comparison to the NLMS predictor.

Chapter 4

Evaluation and Results

In this chapter, the proposed E-NLMS playout buffer algorithm is evaluated in comparison to the basic NLMS playout algorithm. Both playout buffer algorithms are simulated for several network delay traces. Packets are stretched and compressed using the single packet WSOLA-based time-scale modification algorithm. Performance metrics used to evaluate the algorithms are the packet loss rate due to late arrivals, the average end-to-end delay, and the percentage of packets that are scaled using the packet-based WSOLA time-scale modification technique.

Chapter 4 is structured as follows. The method of evaluation for the proposed playout buffer algorithm is described in Section 4.1. Section 4.2 details the various issues involved with end-to-end network delay traces. Accurate measurement of one-way network delays is affected by clock synchronization issues which are detailed in Section 4.2.1. Section 4.2.2 describes the process used to gather the network delay traces. The measured traces are then analyzed in Section 4.2.3. Simulations and experimental results evaluating the E-NLMS algorithm for the various network delay traces are presented and discussed in Section 4.3.

4.1 Method of Evaluation

The proposed E-NLMS algorithm is evaluated by comparing its performance to the original NLMS playout buffer algorithm. A framework is constructed to simulate the different playout buffer algorithms using a set of network delay traces. The playout algorithms are evaluated using the following criteria:

1. Average Playout Delay: The end-to-end playout delay is calculated for each packet and then averaged over the duration of the call.

2. Packet Loss Rate (PLR): The packet loss rate due to late arrivals at the destination is computed. A packet is considered to be 'lost' if it never arrives, or if it arrives after it was scheduled to be played out at the receiver.

3. Percentage of Packets Scaled: The percentage of packets that are time-scaled (stretched or compressed) by the receiver, is computed. The current packet is stretched by the receiver if it is scheduled to finish playing out before the predicted arrival of the next packet. Similarly, packets may be compressed if the playout buffer has become too large and future packets have already arrived.

To evaluate the playout buffer algorithms, end-to-end network delay measurements are collected. The behaviour of the playout delay adjustment algorithms is simulated with the network delay traces. Each algorithm estimates the playout delay, D_i, of the i-th packet. The estimated playout delay is then compared to the actual network delay found in the trace. If the actual delay is larger than the estimated playout delay, the packet will arrive too late to be played out by the playout scheduling algorithm and will be considered 'lost'. At the end of the trace, the average playout delay and the packet loss rate are calculated.

The playout buffer algorithms are also evaluated in conjunction with the single packet WSOLA-based time-scale modification algorithm. The audio packets are stretched or compressed using dynamic time-scale modification, such that the i-th packet finishes playing out just as packet $(i+1)$ is expected to begin playout at the destination. The performance metrics in this case are the average playout end-to-end delay, the packet loss rate, and the percentage of packets that are scaled.

4.2 Network Delay Traces

In order to compare and evaluate the performance of the playout delay adjustment algorithms, network delay traces are needed to simulate the behaviour of the algorithms under identical network conditions. Ideally, one-way network delay traces should be used. However, it is difficult to accurately determine the one-way end-to-end network delay between two nodes due to clock synchronization issues. Therefore, a set of round-trip network delay

traces is collected instead and used in the evaluation of the proposed E-NLMS playout buffer algorithm.

4.2.1 Clock Synchronization

The measurement of one-way delays between two nodes are affected by the following clock synchronization problems [51].

- Clock Offset: The offset is the difference between the clock at the sender and the clock at the receiver at a particular time.
- Clock Skew: Sender and receiver clocks may not be running at the same frequency. The skew is the difference between the two clock frequencies.

The synchronization between the two clocks impacts the accuracy of the measurement. In the case of a unidirectional delay, the sender timestamps the audio packet when it leaves the sender, and the receiver notes the time the packet arrives at the receiver. If the two hosts are perfectly synchronized, the difference between the two timestamps is the end-to-end network delay experienced by the packet. If the clocks have a non-zero offset, the difference in timestamps includes not only the end-to-end packet delay, but also the clock offset. Given a one-way delay measurement, the clock offset cannot be distinguished from the actual end-to-end network delay, unless the actual network delay is known [52]. However, the actual end-to-end network delay is what one is attempting to measure in the first place. If the clocks have relative skew, not only is the end-to-end delay measurement inaccurate due to the offset, but the offset also gradually increases or decreases over time depending on whether the transmitter clock runs slower or faster than the receiver clock [52].

Network Time Protocol (NTP)

The Network Time Protocol (NTP) has been developed to synchronize clocks and coordinate time distribution within the Internet. NTP is used by Internet time severs and their clients to synchronize clocks, as well as automatically organize and maintain the time synchronization subnet itself [53]. The current formal protocol specification is described in RFC 1305 [54], and a detailed description of the architecture, protocol and algorithms used in NTP is found in [55].

The NTP system consists of a network of primary and secondary time servers, clients, and interconnecting transmission paths. A primary time server is directly synchronized to a primary reference source, usually a timecode receiver or calibrated atomic clock [55]. Secondary time servers synchronize themselves to the primary servers. Timestamps are exchanged between NTP servers and clients to determine individual round-trip delays and clock offsets, as well as provide reliable error estimates.

Fig. 4.1 shows how the timestamps are numbered and exchanged between server A and peer B [55]. A packet with timestamp T_1 is sent from server A to peer B where the received timestamp is noted to be T_2. Peer B immediately processes the packet and adds the timestamp T_3 before sending it back to server A where it is received at timestamp T_4. Let $a = T_2 - T_1$ and $b = T_3 - T_4$. Assume that the network is free of congestion, and the network delays, a and b, correspond to the minimum propagation delays between server A and peer B. Then the round-trip delay, δ, and clock offset, θ, of B relative to A at time T_4 are given by

$$\delta = a - b \qquad \text{and} \qquad \theta = \frac{a+b}{2}. \tag{4.1}$$

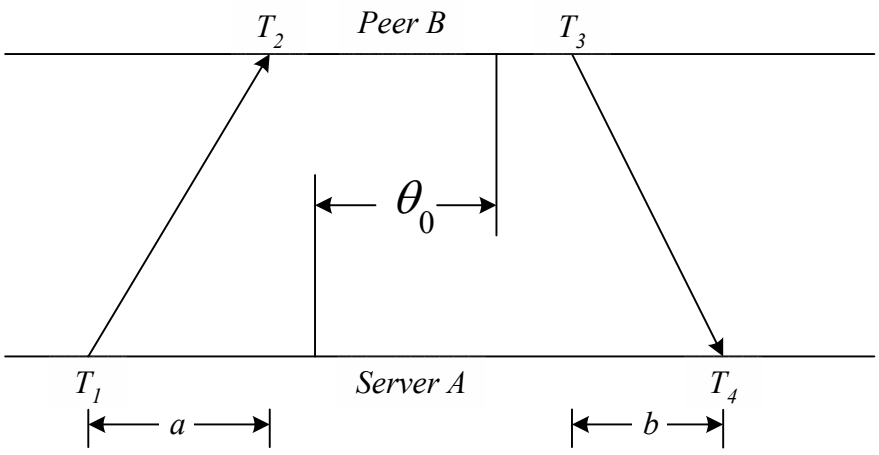


Fig. 4.1 Network Time Protocol [55]

In practice, the network delays will vary due to varying levels of congestion in the Internet. The true offset of B relative to A is called θ_0 in Fig. 4.1. Let x denote the actual delay between the departure of a message from A and its arrival at B. Then, $x + \theta_0 = T_2 - T_1 \equiv a$. Since x must be positive, $x = a - \theta_0 \geq 0$ which requires that $\theta_0 \leq a$. Similarly, it can be seen that $b \leq \theta_0$, thus $b \leq \theta_0 \leq a$. The inequality can be expanded

further and expressed as

$$b = \frac{a+b}{2} - \frac{a-b}{2} \leq \theta_0 \leq \frac{a+b}{2} + \frac{a-b}{2} = a, \tag{4.2}$$

which is equivalent to

$$\theta - \frac{\delta}{2} \leq \theta_0 \leq \theta + \frac{\delta}{2}. \tag{4.3}$$

Therefore, the true clock offset lies in the interval of size equal to the measured delay and centred about the measured offset [55].

The NTP algorithm must produce an accurate estimate of the round trip delay $\hat{\delta}$ and the clock offset $\hat{\theta}$ from a set of samples (δ_i, θ_i) collected for the path under normal traffic conditions. A *minimum filter* is designed which selects from the n most recent samples (δ_i, θ_i), the sample with the lowest round-trip delay, δ_j, and then uses (δ_j, θ_j) as the estimate of the minimum round-trip propagation delay, δ_0 and true clock offset, θ_0, respectively [55].

NTP continuously corrects the time and frequency of the local clock to agree with the time at the synchronization sources. The local clock and its adjustment mechanism are modelled as a *disciplined oscillator* [53]. The time and frequency of the local clock are adjusted via a feedback phase-lock loop (PLL) in order to reduce the clock offset θ_0.

While NTP provides an effective clock synchronization system within the Internet, its performance does not meet the accuracy required of one-way network delay measurements. NTP's accuracy depends in part on the delays of the Internet paths used by NTP peers, and this is precisely what is being measured. NTP works well if paths between nodes are symmetric, however it has been shown that network delays within the Internet are asymmetric [56]. NTP focuses on time accuracy over long time scales (hours to days), which may come at the expense of short-term clock skew and drift [51]. NTP provides accuracy in the order of ten milliseconds [57]; however it does not assure synchronization on the small time scales of individual network delays [58].

Global Positioning System (GPS)

The Global Positioning System (GPS) is primarily used as a navigation system and is a reliable source of positioning for both the military and civilian communities [59]. However, GPS is also globally used to distribute time to a large amount of users and synchronize clocks over large distances with a high degree of precision and accuracy. Time synchroniza-

tion in the Internet can be assured by having local clocks directly derive their time from a GPS receiver [51]. GPS achieves clock synchronization to less than 100 μs [60].

The GPS satellite constellation includes at least 24 satellites orbiting the earth at a height of 20,200 km. GPS satellites broadcast on two carrier frequencies, L1 and L2. Each satellite broadcasts a spread-spectrum waveform, called a pseudo-random noise (PRN) code on L1 and L2. GPS receivers then use the coarse acquisition (C/A) code broadcast on the L1 frequency as their time and frequency reference [61]. The output time from the GPS receiver is thus synchronized to within 100 ns of Coordinated Universal Time (UTC) [62].

Unidirectional network delays within the Internet can be measured accurately by equipping both endpoints with a GPS satellite transceiver. GPS receivers need a direct line-of-sight to the GPS satellite in order to receive the signal, which may not necessarily be possible indoors. The GPS solution also does not scale easily. Network delay traces are desired for additional hosts, and the additional hosts would also need to be equipped with GPS receivers.

Clock Skew Estimation

Clock skew occurs when sender and receiver clocks run at different frequencies, resulting in the clocks to be offset from each other. If the receiver clock runs faster than the sender clock, the clock offset between them will grow and the one-way delay measurements will gradually appear to increase. The measurements may lead one to infer that the network delays are increasing due to greater network congestion and queueing delay, when in fact the increase is due to the difference in sender and receiver clock frequencies.

Clock skew estimation algorithms have been developed to estimate and remove the skew from actual one-way Internet delay traces [58, 63]. While the algorithms are robust at detecting and removing linear skew, they do not necessarily work for non-linear skew [52], nor do they remove any clock offsets that may remain [64].

Round Trip Delays

Measuring round-trip network delays and dividing the results by two to determine the one-way delays avoids the clock synchronization issues discussed above. This method implicitly assumes that Internet paths between two nodes are symmetric, however it has been shown that in most cases, paths between two nodes are in fact asymmetric [65].

Given that the techniques to measure unidirectional network delays are either unreliable or difficult to scale, traces of round-trip delays are collected instead. Since only one clock is used to timestamp packets, round-trip delay measurements avoid any problems associated with clock synchronization. While the round-trip measurements cannot be used directly to determine the one-way delay [56], they give a good idea of the magnitude of the actual one-way network delay and can be used to approximate trends in Internet packet delay.

4.2.2 Collection of Traces

The Internet *ping* service is commonly used to measure round-trip delays. The basic ping program timestamps an ICMP echo packet [66] with the local system time before sending it out onto the Internet. The receiver echoes a reply back to the sender where it is timestamped again. The round-trip delay is the difference between the two local system timestamps. The system time is the time elapsed since the operating system has started, and is not an indication of the actual current time.

Normally, ping waits for the return of each ICMP packet before sending out the next one. The basic ping program is modified here to continuously send a 45 byte ICMP packet every 10 ms. The inter-departure time can be adjusted as desired.

A set of 18 network delay traces was collected between nodes in Canada. Each trace lasts for 2 hours and 47 minutes, and during this time, 1,000,000 packets are transmitted towards the destination. The traces were collected both during the day and at night.

Table 4.1 Network delay traces: Round-trip packets sent from a PC at McGill University

Trace	Date	Start Time	End Time	Sender	Receiver
UBCday	2002-03-28	10:25	13:12	McGill	British Columbia
UBCnight	2002-03-27	22:02	00:49	McGill	British Columbia
UT2day	2002-03-28	13:47	16:34	McGill	Toronto
UT2night	2002-03-28	01:31	04:18	McGill	Toronto
UTday	2002-04-04	12:14	15:02	McGill	Toronto
UTnight	2002-03-27	00:31	03:18	McGill	Toronto
YorkDay	2002-04-09	12:22	15:09	McGill	York
YorkNight	2002-04-09	03:15	06:02	McGill	York

The traces can be divided into two categories. Table 4.1 lists the traces that were gathered from the Telecommunications & Signal Processing Laboratory at McGill Univer-

sity. In this case, the receivers were selected to be at different universities across Canada. Traditionally, network traces have always been gathered between universities and/or other research institutes. The Internet started as a network linking different universities and academic institutions. The universities are directly connected to the backbone of the Internet and therefore, significant bandwidth is available to them. As the ultimate goal of VoIP is to be used in the home environment, network delay traces were also collected between individual homes with high-speed cable access to the Internet. Table 4.2 lists the delay traces that were gathered from a home PC in Montreal. The destination nodes in the different cities were also connected via cable modem to the Internet.

Table 4.2 Network delay traces: Round-trip packets sent from a home PC in Montreal connected to the Internet by cable modem

Trace	Date	Start Time	End Time	Sender	Receiver
Mtl2Night	2002-04-11	02:45	05:31	Montreal	Montreal
MtlNight	2002-04-09	02:56	05:42	Montreal	Montreal
Ott2Day	2002-04-12	11:21	14:08	Montreal	Ottawa
Ott2Night	2002-04-12	02:41	05:27	Montreal	Ottawa
OttDay	2002-03-27	10:50	13:37	Montreal	Ottawa
OttNight	2002-03-27	02:02	04:49	Montreal	Ottawa
TO2day	2002-03-28	10:34	13:21	Montreal	Toronto
TO2night	2002-03-28	02:41	05:27	Montreal	Toronto
TOday	2002-03-22	10:33	13:20	Montreal	Toronto
TOnight	2002-03-22	03:09	05:56	Montreal	Toronto

Network Delay Traces Obtained From Other Sources

In addition to generating network delay traces, the traffic traces used in [12] were acquired. Table 4.3 lists the 12 one-way network delay traces obtained. The 12 one-way delay traces were recorded for streams traversing the Internet to AT&T in New Jersey from four different hosts, three in the US (Carnegie-Mellon University, Rutgers University, and Stanford University), and one host in the UK (Cambridge University). Each trace lasted 10 minutes and the traces are captured at different times of the day.

UDP packets [18] are generated by the sender and transmitted to the receiver. Each UDP packet carried 20 ms of audio and contained a sequence number and sender timestamp [12]. At the receiver, the arrival time was recorded and the difference in timestamps

Table 4.3 Network delay traces obtained from other sources and used in [12]: One-way packets received at AT&T (NJ)

Trace	Date	Start Time	End Time	Sender	Receiver
c1	1995-08-02	10:00	10:10	Cambridge	AT&T (NJ)
c3	1995-08-01	16:00	16:10	Cambridge	AT&T (NJ)
c5	1995-08-01	20:00	20:10	Cambridge	AT&T (NJ)
m1	1995-07-27	09:00	09:10	Carnegie-Mellon	AT&T (NJ)
m3	1995-07-31	15:00	15:10	Carnegie-Mellon	AT&T (NJ)
m5	1995-07-26	21:00	21:10	Carnegie-Mellon	AT&T (NJ)
r1	1995-07-27	08:00	08:10	Rutgers	AT&T (NJ)
r3	1995-07-31	16:00	16:10	Rutgers	AT&T (NJ)
r5	1995-07-26	20:00	20:10	Rutgers	AT&T (NJ)
s1	1995-07-27	10:00	10:10	Stanford	AT&T (NJ)
s3	1995-08-01	14:00	14:10	Stanford	AT&T (NJ)
s5	1995-07-26	22:00	22:10	Stanford	AT&T (NJ)

gives an estimate of the network delay variation. As the sender and receiver clocks were not synchronized, the difference in timestamps includes the clock offset. The network delay was thus defined to be the quantity in excess of the minimum observed delay from sender to receiver for a given stream [12]. The network delay values are therefore computed by finding the packet with the smallest difference between sender and receiver timestamps, and subtracting that minimum delay value from each network delay. Clock drift is assumed to be negligible over the 10 minute lifetime of each trace.

4.2.3 Analysis of Measured Traces

The collected round-trip delay traces are graphed and analyzed. Figs. 4.2, 4.3, 4.4, and 4.5 display trace segments representative of the collected network delay traces. For each trace, the round-trip delay in milliseconds is plotted as a function of the packet number. A round-trip delay value of 0 ms indicates that the packet never returned and is lost.

Figs. 4.2 and 4.3 display round-trip traces taken from McGill University to other universities within Canada. Both the sender and receiver are directly connected to the Internet. Fig. 4.2 plots a trace collected during the day between McGill University and the University of British Columbia, while a trace collected at night between McGill and York Universities, is shown in Fig. 4.3.

Traces were also taken between individual homes with high-speed access to the Internet.

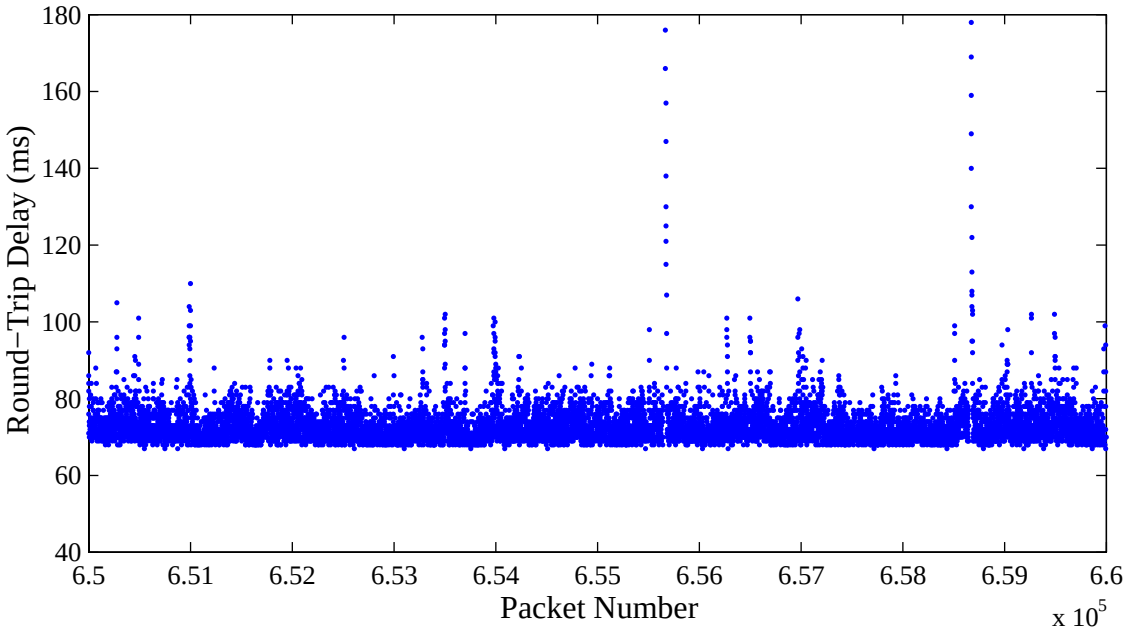


Fig. 4.2 Round-trip times (RTT) for trace 'UBCday' from McGill Univ. (Montreal) to Univ. of British Columbia (Vancouver)

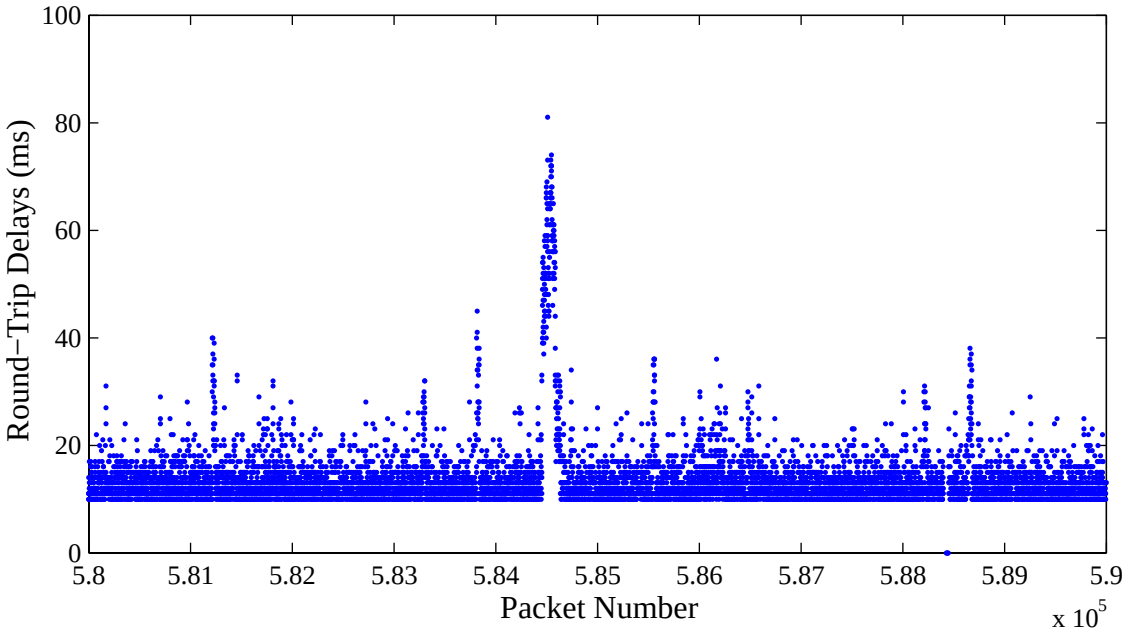


Fig. 4.3 Round-trip times (RTT) for trace 'YorkNight' from McGill Univ. (Montreal) to York Univ. (Toronto)

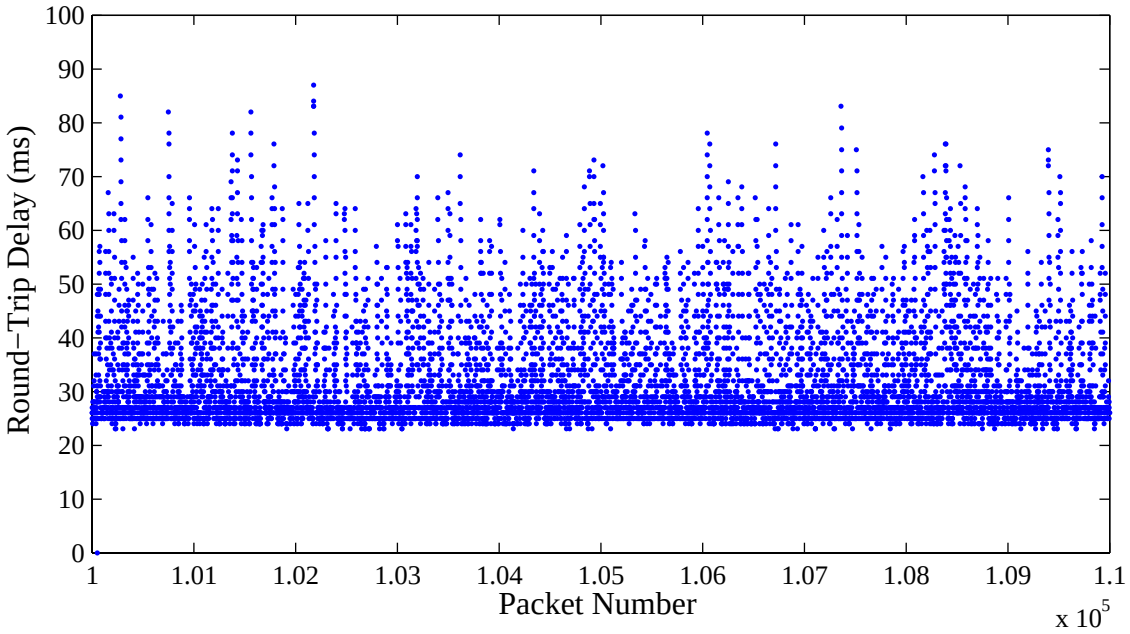


Fig. 4.4 Round-trip times (RTT) for trace 'TO2day' from Montreal (Videotron cable) to Toronto (Rogers cable)

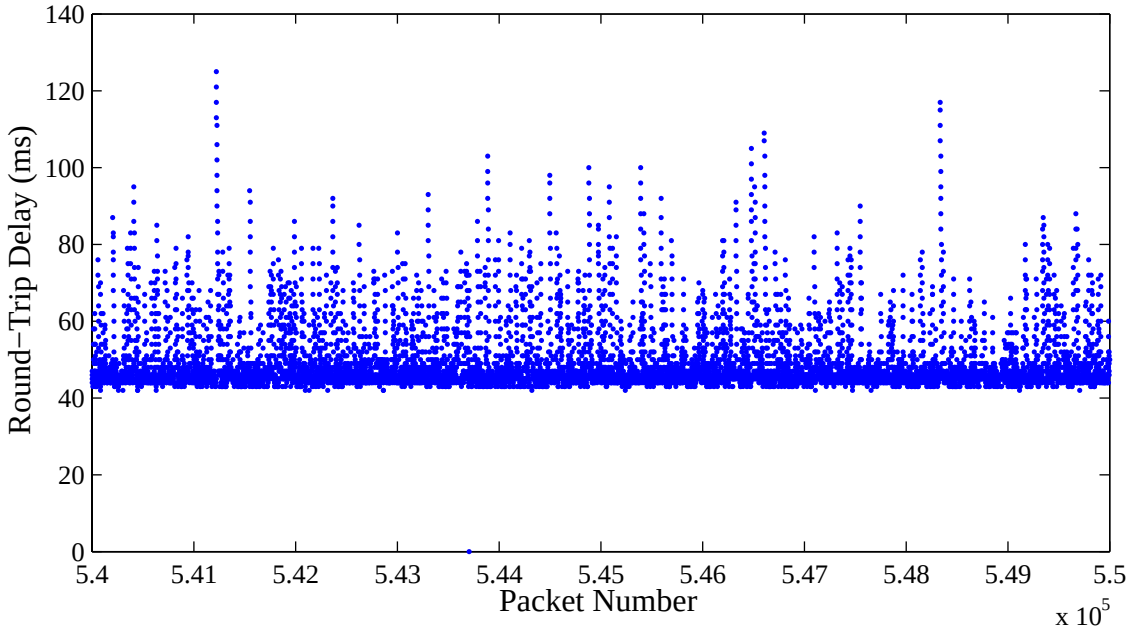


Fig. 4.5 Round-trip times (RTT) for trace 'OttNight' from Montreal (Videotron cable) to Ottawa (Rogers cable)

Figs. 4.4 and 4.5 display round-trip traces taken between a computer in a Montreal home and computers at homes in Toronto and Ottawa, respectively. The home computers are connected to the Internet via cable modem. Fig. 4.4 displays a trace collected during the day between homes in Montreal and Toronto, and a trace collected at night between homes in Montreal and Ottawa is graphed in Fig. 4.5.

While each individual delay trace is unique and cannot be scrutinized on its own, the round-trip delay traces are representative of delays in the Internet, and can be analyzed for common trends. The measured network traces are found to contain frequent occurrences of network delay spikes and there is great variation in the packet round-trip delay values. From the different traces plotted in Figs. 4.2–4.5, it can be seen that the round-trip times vary considerably over both small and large time scales. For each trace, the minimum, median, mean, and maximum network delays are determined. The network delay values that are greater than 90% and 99% of the delays in the trace, are also calculated. As well, the standard deviation of the network delays and the packet loss rate (PLR) are also computed. Tables 4.4 and 4.5 summarize the statistics for the two categories of network delay traces collected.

Table 4.4 Statistics for round-trip delay traces from McGill University

Trace	PLR (%)	Min. (ms)	Median (ms)	90% (ms)	99% (ms)	Max. (ms)	Mean (ms)	Std. Dev. (ms)
UBCday	0.04	67	70	75	92	569	72.2	20.4
UBCnight	0.06	67	74	84	137	727	78.5	33.7
UT2day	0.34	10	11	20	286	1035	21.9	52.9
UT2night	0.32	9	10	13	68	556	12.4	15.0
UTday	0.05	9	12	46	334	2110	27.6	59.4
UTnight	0.04	9	10	13	184	662	16.3	32.7
YorkDay	0.15	10	22	104	152	2161	40.9	41.8
YorkNight	0.01	10	12	17	28	991	13.0	6.4

4.3 Simulations and Experimental Results

The behaviour of both the NLMS and E-NLMS algorithms are simulated with the gathered network delay traces. In the first experiment, the NLMS and E-NLMS algorithms compute and estimate the end-to-end playout delay, D_i, of packet i. If the estimated playout delay

Table 4.5 Statistics for round-trip delay traces from a home computer in Montreal

Trace	PLR (%)	Min. (ms)	Median (ms)	90% (ms)	99% (ms)	Max. (ms)	Mean (ms)	Std. Dev. (ms)
Mtl2Night	0.06	12	17	37	258	585	22.0	17.8
MtlNight	0.04	12	18	35	60	473	27.4	38.3
Ott2Day	0.02	27	35	57	77	206	40.0	11.4
Ott2Night	0.01	26	32	47	67	196	35.0	8.3
OttDay	0.08	40	48	68	88	1648	53.1	23.3
OttNight	0.01	40	46	61	82	236	49.4	8.5
TO2day	0.01	20	28	47	67	285	32.1	10.1
TO2night	0.02	20	26	44	63	1387	31.1	15.0
TOday	0.03	21	28	48	68	526	33.0	11.6
TOnight	0.01	20	27	44	63	1505	31.1	18.1

is larger than the actual network delay found in the delay trace, the packet is assumed to arrive in time to be played out. In the opposite case, if the estimate of the end-to-end delay is lower than the actual delay value, the packet will arrive too late to be played out and will be counted as 'lost'. At the end of the experiment, the average end-to-end delay and the packet loss rate are computed.

The second experiment requires the implementation of the single packet WSOLA-based time-scale modification algorithm. As in the first experiment, the NLMS and E-NLMS algorithms are used to estimate the end-to-end delay of the next packet. The current packet i is then stretched or compressed if necessary, so that the current packet just finishes playing out as packet $(i + 1)$ is scheduled to begin playout. For this experiment, the performance metrics used to evaluate the playout algorithms are the average end-to-end playout delay, the loss rate due to late packet arrivals, and the percentage of packets that are time-scaled (stretched or compressed).

4.3.1 Playout Delay Estimation

The NLMS and E-NLMS algorithms are simulated offline for the collected network delay traces. Both algorithms use the past network delay values to predict the current packet's network delay, $\hat{d}_i$, and compute its end-to-end delay, D_i. The end-to-end delay value, D_i, is then compared to the actual network delay value, n_i, recorded in the trace to determine if the packet arrived in time to be played out. The average end-to-end delay of all packets

and the packet loss rate due to late packet arrivals are computed at the end of the trace.

Table 4.6 NLMS and E-NLMS algorithm parameter values

Parameter	*Value*
$\mathbf{w}_0$	$[1\ 0\ \ldots\ 0]^T$
N	20
μ	0.001

Table 4.6 lists the empirical values used for the parameters of the NLMS and E-NLMS algorithms. For the two experiments in [12], the value of the safety factor, β, was set to 4.0 and 6.0, respectively. The value of β is varied from 4.0 to 6.0 in this simulation, in order to illustrate the tradeoff between average end-to-end delay and late packet 'loss'. The results are plotted in Figs. 4.6–4.8 for three of the traces.

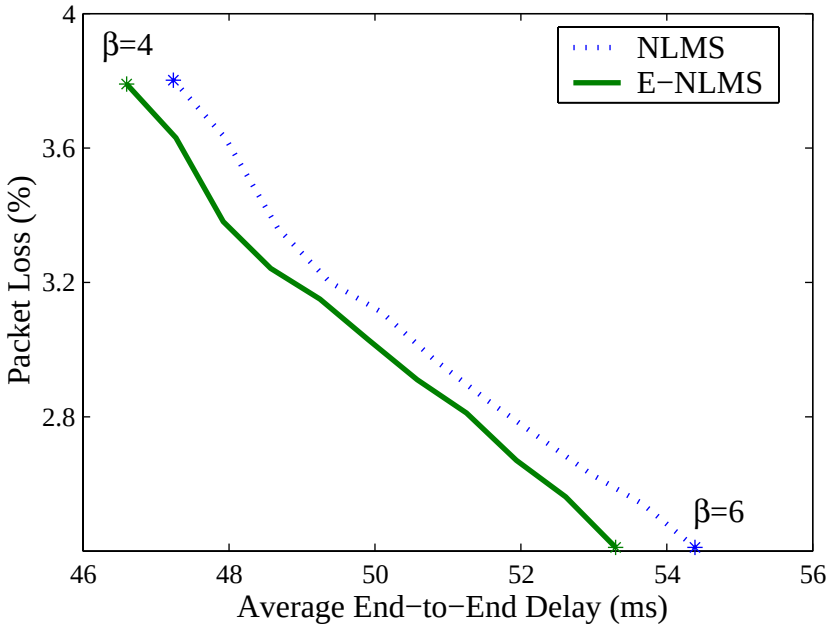


Fig. 4.6 Tradeoff between 'late' packet loss and average end-to-end packet delay for trace 'TO2day'

Figs. 4.6–4.8 show that both the average end-to-end delay and 'late' packet loss are reduced for the E-NLMS algorithm as compared to the original NLMS predictor. An improvement in overall performance is achieved by incorporating a spike mode into the basic NLMS algorithm and reducing the delay overestimation during spikes.

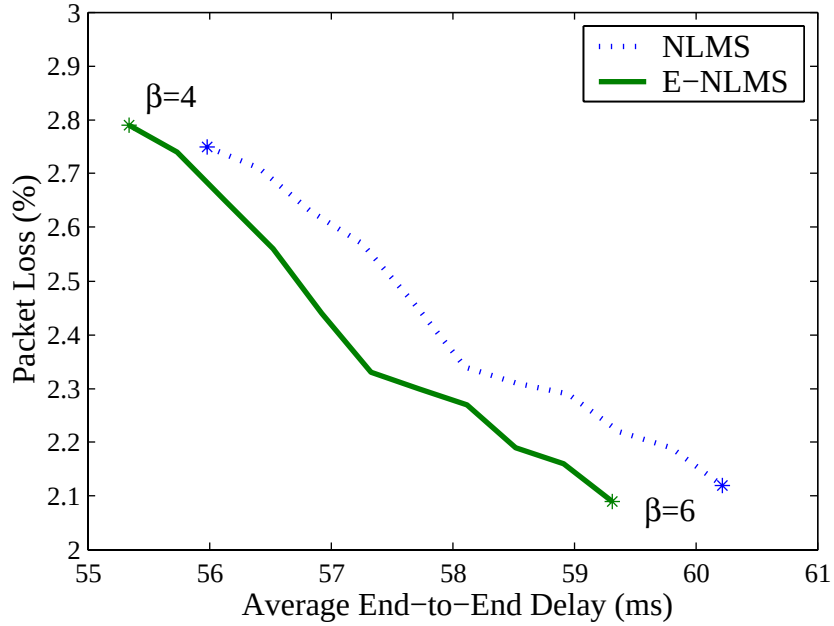


Fig. 4.7 Tradeoff between 'late' packet loss and average end-to-end packet delay for trace 'OttNight'

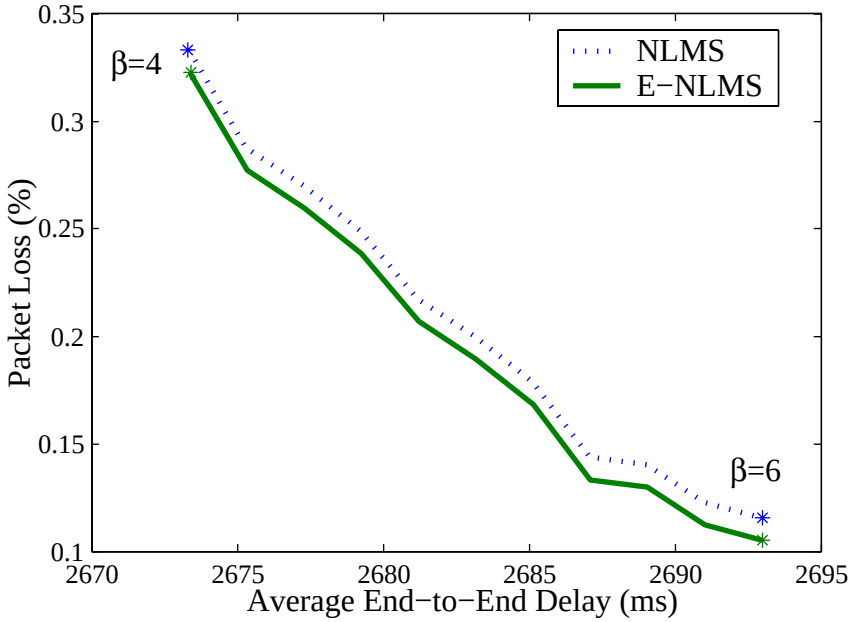


Fig. 4.8 Tradeoff between 'late' packet loss and average end-to-end packet delay for trace 'm5' used in [12]

Table 4.7 Experiment 1: Comparison of NLMS and E-NLMS algorithms. Note that the variation in network delay, v_N, computed at the end of the trace, as calculated by Eq. (3.5) is also given. The E-NLMS algorithm outperforms the NLMS predictor when variation in network delay is higher. The top group of traces is collected between universities; the middle group of traces is collected between home PC's connected via cable to the Internet; the bottom group of traces is obtained from other sources and was used in [12].

Trace	v_N	Better Algorithm
UBCday	2.6	NLMS
UBCnight	3.2	NLMS
UT2day	3.4	NLMS
UT2night	1.3	NLMS
UTday	3.4	NLMS
UTnight	3.7	NLMS
YorkDay	3.7	NLMS
YorkNight	1.9	NLMS
Mtl2Night	4.1	E-NLMS
MtlNight	4.8	E-NLMS
Ott2Day	4.1	E-NLMS
Ott2Night	3.8	E-NLMS
OttDay	4.0	E-NLMS
OttNight	3.7	E-NLMS
TO2day	4.2	E-NLMS
TO2night	2.5	E-NLMS
TOday	4.3	E-NLMS
TOnight	4.0	E-NLMS
c1	9.3	E-NLMS
c3	9.9	E-NLMS
c5	9.0	E-NLMS
m1	10.3	E-NLMS
m3	12.1	E-NLMS
m5	10.2	E-NLMS

The NLMS and E-NLMS algorithms are simulated and tested for all of the traces. The value of β is varied between 4.0 and 6.0 in each case, and the packet loss rate is plotted versus the average end-to-end delay for each trace. In general, it was found that the E-NLMS algorithm outperforms the NLMS predictor, by reducing both the average end-to-end delay and the packet loss rate for a given trace. The qualitative results for all the traces are summarized in Table 4.7.

As seen in Table 4.7, the E-NLMS algorithm outperforms the original NLMS predictor for a majority of the 24 traces tested. The variation in network delay is found to be lower for the traces where the E-NLMS algorithm did not outperform the NLMS predictor. The E-NLMS algorithm detects a spike when either the previous packet was lost, or the network delay estimate, $\hat{d}_i$, exceeds the actual network delay, n_i, by a threshold. In this case, the threshold is set to $5v_i$. When the variation in network delay, v_i, is low, the E-NLMS algorithm will consider a small increase in network delay to be a delay spike and thus switch to spike mode. While in spike mode, the safety buffer term, $\beta\hat{v}_i$, is reduced by a factor of 4. Thus, the E-NLMS algorithm will closely track the network delay and will miss any subsequent packets that have a small increase in network delay. The E-NLMS predictor can be further improved by enhancing the spike detection algorithm.

Table 4.7 also shows that the E-NLMS algorithm outperforms the NLMS predictor for all traces collected between individual homes with high-speed access to the Internet. Network delays between individual homes with cable access are generally higher and incur greater variation from packet to packet. To enable VoIP to be used by individuals at home, the average end-to-end delay and packet loss must be reduced. The E-NLMS playout buffer algorithm achieves an overall improvement in performance by reducing both the average end-to-end delay and packet loss for the traces collected between homes with high-speed access to the Internet.

4.3.2 Playout Scheduling With Dynamic Time-Scale Modification

The single packet WSOLA-based time-scale modification algorithm is implemented for the second experiment. As in the first experiment, the NLMS and E-NLMS algorithms are both used to predict the next packet's network delay, $\hat{d}_{i+1}$, and compute its end-to-end playout delay, D_{i+1}. The current audio packet i is then scaled using the dynamic time-scale modification algorithm such that the current packet finishes playing out just as the

next packet $(i + 1)$ is scheduled to begin playout.

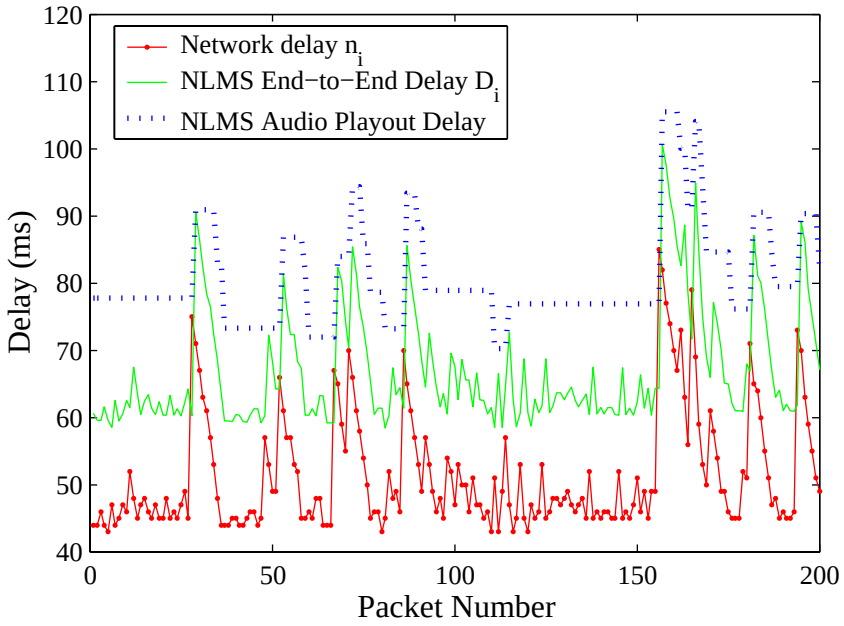


Fig. 4.9 End-to-end packet delay and actual audio playout delay for trace 'OttDay' using the NLMS algorithm

The NLMS and E-NLMS algorithm parameter values are set to the same values used in Experiment 1 and shown in Table 4.6. The value of the safety factor, β, is set to 4. The NLMS and E-NLMS algorithms are simulated by mapping a 40 minute speech file to the various network delay traces. The algorithms estimate and compute the end-to-end delay for each audio packet. If the current packet i is expected to finish playing out before the predicted arrival time of packet $(i + 1)$, the current speech packet is stretched so that it finishes playing out just as the next one is expected to arrive. However, in order to avoid constant scaling, the playout buffer is allowed to build up, and audio packets are only compressed when the next packet $(i + 1)$, is predicted to arrive before the current packet i, will begin playing out.

Figs. 4.9 and 4.10 display the actual network delay incurred by packets, the end-to-end packet delay computed by the playout buffering algorithm, and the actual audio playout delay, after single packet WSOLA-based time scale modification is used to stretch or compress the audio packets, for both the NLMS and E-NLMS algorithms, respectively.

Both the NLMS and E-NLMS playout scheduling algorithms achieve low packet loss and

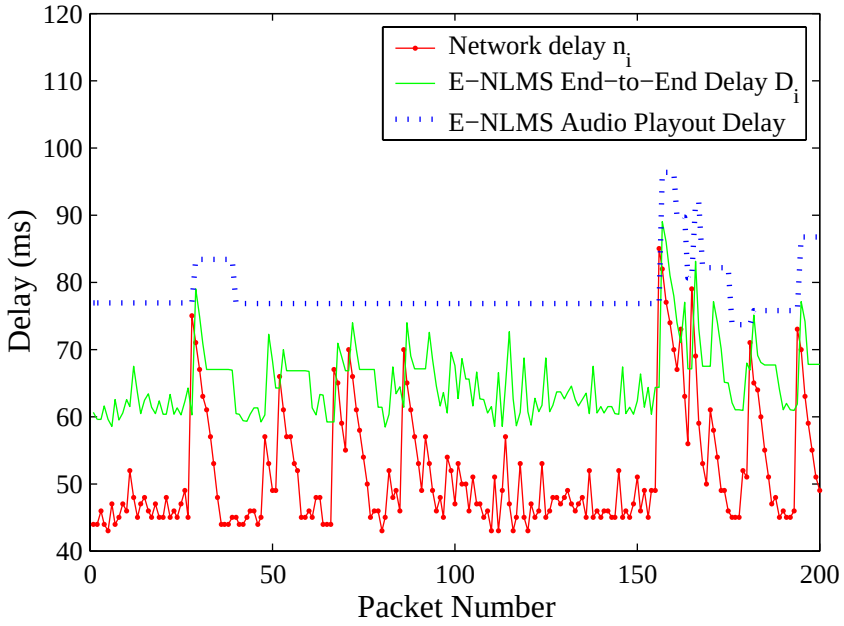


Fig. 4.10 End-to-end packet delay and actual audio playout delay for trace 'OttDay' using the E-NLMS algorithm

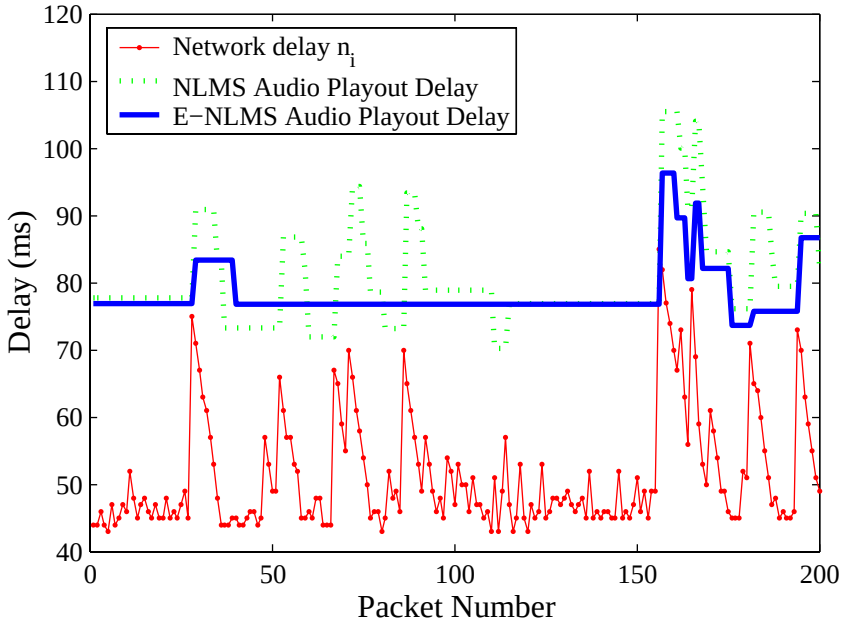


Fig. 4.11 Actual audio playout delay for trace 'OttDay' using both NLMS and E-NLMS algorithms

low end-to-end delay as seen in Figs. 4.9 and 4.10. The basic NLMS algorithm reacts very rapidly to spikes and this results in greater time-scaling of packets. The E-NLMS algorithm, on the other hand, computes a lower end-to-end delay than the NLMS predictor during spikes, and thus does not stretch or compress packets as often as the NLMS algorithm.

Fig. 4.11 plots the actual audio playout delay obtained after dynamic time-scale modification using both the NLMS and E-NLMS playout algorithms. For a given value of β, the E-NLMS playout buffer algorithm generally achieves lower playout delay while maintaining almost the same packet loss rate. More importantly, the E-NLMS algorithm results in considerably less time-scaling of packets as compared to NLMS, as shown in Fig. 4.11.

The average playout delay, the packet loss rate (PLR), and the percentage of packets stretched and compressed using packet-based WSOLA time-scale modification, are computed using both the NLMS and E-NLMS algorithms. Tables 4.8 and 4.9 summarize the results for all traces. The one-way traces used in [12] last for only 10 minutes and are too short to be used in this experiment.

Table 4.8 Experiment 2: Comparison of NLMS and E-NLMS algorithms with dynamic time-scale modification for traces between universities. Average playout delay (Avg. delay), packet loss rate (PLR), percentage of packets stretched (% str.), and percentage of packets compressed (% comp.) are computed.

Trace	NLMS				E-NLMS			
	Avg. delay (ms)	PLR (%)	% str.	% comp.	Avg. delay (ms)	PLR (%)	% str.	% comp.
UBCday	96.0	0.14	0.62	1.14	95.7	0.16	0.38	0.67
UBCnight	118.0	0.11	2.73	5.27	117.9	0.14	2.56	4.90
UT2day	53.0	0.74	1.37	8.43	52.5	0.93	1.09	7.65
UT2night	35.7	0.43	0.55	2.94	35.7	0.49	0.44	2.70
UTday	71.9	0.95	2.89	14.55	71.0	1.16	2.62	12.93
UTnight	47.3	0.63	1.08	6.65	46.7	0.75	1.03	6.29
YorkDay	88.8	0.15	7.98	16.10	88.6	0.20	7.74	16.07
YorkNight	37.9	0.03	0.33	0.58	37.8	0.05	0.15	0.32

The E-NLMS algorithm reduces the playout delay while maintaining the same packet 'loss' rate for all the traces in Tables 4.8 and 4.9. As well, using the E-NLMS playout scheduling algorithm results in substantially fewer packets being time-scaled (stretched or compressed). The packet-based WSOLA time-scale modification algorithm is implemented

Table 4.9 Experiment 2: Comparison of NLMS and E-NLMS algorithms with dynamic time-scale modification for traces between home PCs. Average playout delay (Avg. delay), packet loss rate (PLR), percentage of packets stretched (% str.), and percentage of packets compressed (% comp.) are computed.

	NLMS				E-NLMS			
Trace	Avg. delay (ms)	PLR (%)	% str.	% comp.	Avg. delay (ms)	PLR (%)	% str.	% comp.
Mtl2Night	56.1	0.83	4.88	11.95	54.3	0.92	2.85	5.63
MtlNight	57.9	0.84	5.44	13.42	56.0	0.92	3.70	6.93
Ott2Day	71.8	1.44	5.02	14.36	69.4	1.39	3.12	6.65
Ott2Night	65.2	1.27	3.80	10.37	63.6	1.25	1.96	4.52
OttDay	82.4	1.18	4.57	12.01	80.3	1.18	2.35	4.92
OttNight	83.2	1.25	4.66	12.39	80.9	1.27	2.51	5.30
TO2day	61.9	1.27	4.04	10.83	60.1	1.26	2.31	5.04
TO2night	54.2	1.06	2.28	5.84	53.4	1.09	1.38	3.17
TOday	63.9	1.19	4.64	11.92	61.8	1.24	2.53	5.11
TOnight	62.2	1.25	4.03	10.90	60.4	1.26	2.30	4.96

in this work, and listeners are unable to discern any difference in audio quality. The results of the informal listening tests agree with the conclusions presented in [40]. However, it is still preferable to reduce the percentage of packets that are stretched or compressed. As shown in Table 4.9, E-NLMS achieves a more significant improvement in performance for traces between individual homes with high-speed access to the Internet. There is greater fluctuation in network delay for traces between individual homes than for traces between nodes at universities. An increased number of delay spikes leads to a larger variation in network delay, v_i, resulting in the bi-modal E-NLMS algorithm switching modes, and thus outperforming the NLMS predictor.

4.4 Chapter Summary

In this chapter, the proposed E-NLMS playout buffer algorithm was evaluated in comparison to the basic NLMS predictor, for a set of network delay traces. The performance metrics used to evaluate and compare the two playout scheduling algorithms were the packet loss rate due to late packet arrivals, the average end-to-end packet delay, and the percentage of audio packets that were time-scaled.

To simulate the playout scheduling algorithms, a set of network delay traces was required. Ideally, one-way network delay traces are preferred. However, the measurement of one-way delays is difficult over the Internet due to clock synchronization problems. Thus, round-trip delay measurements were collected instead and clock synchronization issues were avoided. While round-trip measurements cannot directly give the one-way network delay, they can be used to determine trends in Internet packet delay. A set of 18 round-trip network delay traces, each lasting for 1,000,000 packets, was collected between nodes in Canada. Additionally, the one-way delay traces used in [12] were also obtained.

Two experiments were conducted to evaluate the NLMS and E-NLMS playout buffer algorithms. In the first experiment, each algorithm was simulated with network delay values from each trace. The algorithms predicted the network delay and computed the end-to-end playout delay for the current packet. The end-to-end packet delay was then compared to the actual network delay value found in the trace to determine if the packet would have arrived in time. At the end of the trace, the packet loss rate due to late arrivals and the average end-to-end packet delay were calculated. The E-NLMS algorithm outperformed the NLMS predictor by reducing both packet loss as well as average end-to-end delay in most cases. In particular, the E-NLMS algorithm provided better performance when the variation in network delay was higher.

For the second experiment, the single packet WSOLA-based time-scale modification algorithm was implemented. The NLMS and E-NLMS playout algorithms were used again to predict the network delay and compute the end-to-end playout delay. The playout delay was then adjusted on a per-packet basis using the packet-based WSOLA time-scale modification technique. The simulations showed that the E-NLMS playout scheduling algorithm reduces the average end-to-end audio playout delay while maintaining approximately the same packet 'loss' rate. More importantly, since the E-NLMS algorithm reduces the safety buffer term as well as the end-to-end delay during spikes, a significantly lower percentage of packets are time-scaled (stretched or compressed) when E-NLMS is used as compared to NLMS. Once again, the E-NLMS algorithm provides a greater improvement in performance for traces where the variation in network delay was higher.

Chapter 5

Conclusion

This thesis has focussed on the design, implementation, and evaluation of an adaptive playout algorithm with delay spike detection for use in real-time Voice over IP (VoIP). An existing playout algorithm, based on the normalized least mean square (NLMS) algorithm, is improved by introducing a spike-detection mode. The proposed algorithm is denoted as the enhanced NLMS (E-NLMS) algorithm. The E-NLMS algorithm computes a more accurate prediction of the network delay during delay spikes and rapidly adjusts the playout delay during such spikes. The E-NLMS algorithm improves overall performance by reducing both the average end-to-end packet delay as well as the packet loss rate, in comparison to the original NLMS predictor.

This chapter summarizes the thesis work. Potential directions for future research are also provided.

5.1 Thesis Summary

The thesis begins by outlining the rapid growth of Internet telephony or VoIP. The circuit-switched Public Switched Telephone Network (PSTN) was designed for real-time voice calls, while the Internet and other packet-based networks were originally created for the efficient transportation of data. Section 1.1 describes how voice calls are carried in the two different networks. The growth in VoIP is being driven by a number of factors including the need to integrate voice and data, bandwidth consolidation, and lower tariffs for voice calls. To succeed in the marketplace, VoIP must offer the same reliability and voice quality as the PSTN. Specifically, high speech signal fidelity, low end-to-end packet delay, low variation

in delay (jitter), and low packet loss are desired for high voice quality in VoIP.

A general background of VoIP is given in Chapter 2. The Internet Protocol (IP) is the packet forwarding protocol used in IP-based packet networks such as the Internet. IP is an unreliable, best-effort, connectionless protocol and does not guarantee a quality of service. Transport layer protocols such as TCP and UDP are used by applications to manage the flow of data between two end hosts. A number of application layer protocols have been developed for Internet telephony. The RTP protocol provides a mechanism for real-time delivery of voice, video, and data. The ITU H.323 and IETF SIP protocols are two of the main protocols used for call signalling in VoIP. The MEGACO/H.248 standard has been developed so that VoIP can interoperate with the PSTN.

The transmission of real-time voice from one endpoint to another in a VoIP system consists of several steps. The analog voice signal is sampled periodically and encoded into a digital stream at the transmitter. Line echoes are removed from the digital signal. A voice activity detector (VAD) is used to detect periods of silence when the user is listening and not talking. The speech signal is divided into frames and then coded using a speech coder. The speech coder may compress the signal in order to reduce the bit rate. Silence suppression or discontinuous transmission (DTX) schemes may also be used to conserve bandwidth. The speech frame is then packetized: first into an RTP packet at the application layer, then into a UDP packet at the transport layer, and finally into an IP packet at the network layer. The IP packet is sent onto the Internet, where it is routed to the destination. Packets may be lost or delayed due to congestion in the network. A playout buffer is used at the receiver to buffer packets and remove *jitter*. The receiver extracts the speech frame from the received packet. The speech frame is decoded with the speech decoder and converted back to analog form to be played out. Packets that do not arrive or arrive after the scheduled playout time, are considered to be 'lost'. Packet loss concealment (PLC) techniques are used to compensate for missing packets.

Chapter 3 introduced the various approaches to reducing jitter or variation in network delay. The role of the playout buffer in receiver-based approaches was discussed and the main types of playout buffer algorithms were reviewed. The three main types of playout buffer algorithms are *autoregressive (AR) estimate-based* algorithms, *statistically-based* algorithms, and *adaptive filter-based* algorithms.

Autoregressive estimate-based algorithms use autoregressive estimates of network delay and jitter to compute the playout delay. Statistically-based approaches determine the

playout delay based on a statistical representation of past delays. Adaptive filter-based approaches aim to predict the network delay by minimizing the mean square error between the actual delay and the estimate. An accurate prediction of the network delay can be used to adjust the playout delay more effectively. The main playout buffer algorithms are not robust enough to adapt their packet delay estimates in the presence of delay spikes, thus spike detection has been incorporated in some playout buffer algorithms.

Playout buffer algorithms have traditionally adjusted their delay estimates during periods of silence by lengthening or shortening the silence between talkspurts. In a more recent approach, the playout delay is adjusted on a per-packet basis by dynamically time-scaling individual voice packets. Packet-based adjustment provides increased flexibility as the playout scheduling algorithm can take advantage of accurate network delay estimates.

The adaptive NLMS predictor uses an accurate prediction of the network delay to closely track network fluctuations. A drawback of the NLMS playout algorithm is that it does not detect delay spikes. Delay spikes are characterized by a large increase in delay followed by a series of declining network delays. Thus, the NLMS prediction is quite large for subsequent network delays during a spike. The proposed playout algorithm, the enhanced NLMS (E-NLMS) algorithm improves the basic NLMS predictor by introducing a spike detection mode to rapidly adjust the playout delay during delay spikes. The safety buffer term can be reduced during spike mode, thereby reducing the overall end-to-end packet delay.

Network delay traces were gathered to simulate and evaluate the E-NLMS playout algorithm. A set of 18 round-trip network delay traces, each lasting for 1,000,000 packets, was collected between hosts in Canada. One-way delay traces used in [12] were also obtained. Two experiments were conducted to evaluate the NLMS and E-NLMS playout buffer algorithms.

In the first experiment, the algorithms predicted the network delay for the current packet and computed its playout delay. The computed packet delay was then compared to the actual network delay value found in the trace to determine whether the packet would have arrived in time. At the end of the trace, the packet loss rate due to late arrivals and the average end-to-end packet delay were calculated. The E-NLMS algorithm outperformed the basic NLMS predictor by reducing both packet loss and average end-to-end delay for most traces. In particular, the E-NLMS algorithm provided better performance when the variation in network delay was higher.

For the second experiment, the single packet WSOLA-based time-scale modification

algorithm was implemented. The NLMS and E-NLMS playout algorithms were used to predict the network delay and compute the end-to-end playout delay. Playout delay adjustment was then performed on a per-packet basis by dynamically time-scaling individual speech packets. In general, the E-NLMS playout scheduling algorithm reduced the average end-to-end audio playout delay while maintaining approximately the same packet 'loss' rate. More importantly, since the E-NLMS algorithm reduces the safety buffer term during spikes, a significantly lower percentage of packets are time-scaled (stretched or compressed) when the E-NLMS algorithm is used as compared to NLMS. Informal listening tests concluded that the degradation due to infrequent time-scaling was inaudible. Once again, the E-NLMS algorithm obtained a greater improvement in performance for traces where the variation in network delay was higher.

5.2 Future Research Work

This section will provide directions for future work on the enhanced NLMS (E-NLMS) algorithm. The main feature of the E-NLMS playout algorithm was to incorporate a spike-detection mode into the basic NLMS predictor. Improving the spike-detection algorithm is a key area for future research. The E-NLMS algorithm uses the past delay values to predict the current network delay. However, if the packet is never received, the E-NLMS algorithm will not be able to update the vector of past delays with a delay value for the current packet. An additional area for further research is how the E-NLMS playout algorithm should update tap weights in the case of a packet that never arrives.

The proposed E-NLMS algorithm provides considerable improvement for delay traces where the variation in delay is high. However, in cases where the variation in delay is low, the E-NLMS algorithm may not necessarily provide any improvement and may in fact perform worse than the existing NLMS predictor.

The current E-NLMS algorithm detects a spike when either the previous packet was 'lost' (arrived too late to be played out) or the delay estimate was greater than the actual delay value by more than a threshold. The threshold is currently set at $5 \times v_i$, where v_i is the estimate of the variation in network delay. For traces with low variation in network delay, the threshold will be quite low and the E-NLMS algorithm will consider a small increase in the network delay to be a spike and will switch to spike mode. In spike mode, the safety buffer term, βv_i, is significantly reduced; thus the end-to-end delay closely tracks

the actual network delay. Thus, even a slight increase in network delay may result in packets arriving late and hence being 'lost' as the delay estimate will be too low. The spike-detection algorithm needs to be improved to only detect legitimate delay spikes.

The E-NLMS algorithm predicts the network delay by passing a vector of past delays through an FIR filter and minimizing the mean square error between the actual delay and the estimate. The algorithm assumes that every packet is received at the destination and has a corresponding network delay value. All playout buffer algorithms make this assumption. There is no mechanism to account for packets that never reach their intended destination, and how the adaptive playout algorithm should react in such a situation. Currently, playout algorithms are evaluated offline with network delay traces. The traces contain only the network delays for packets that arrived at the destination and omit any missing packets. Research needs to be conducted into how adaptive playout algorithms should estimate and compute the playout delay of the current packet, if the previous packet was never received.

References

[1] G. Thomsen and Y. Jani, "Internet telephony: Going like crazy," *IEEE Spectrum*, vol. 37, pp. 52–58, May 2000.

[2] T. J. Kostas, M. S. Borella, I. Sidhu, G. M. Schuster, J. Grabiec, and J. Mahler, "Real-time voice over packet-switched networks," *IEEE Network*, vol. 12, pp. 18–27, Jan.-Feb. 1998.

[3] J. Davidson and J. Peters, *Voice Over IP Fundamentals*. Indianapolis, IN: Cisco Press, 2000.

[4] U. Black, *Voice Over IP*. Upper Saddle River, NJ: Prentice Hall, 2000.

[5] ITU-T Recommendation G.114, "One-way transmission time," May 2000.

[6] R. Ramjee, J. Kurose, D. Towsley, and H. Schulzrinne, "Adaptive playout mechanisms for packetized audio applications in wide area networks," in *Proc. IEEE Infocom Conf. Comp. Commun.*, vol. 2, (Toronto, Canada), pp. 680–688, June 1994.

[7] Y. J. Liang, N. Färber, and B. Girod, "Adaptive playout scheduling and loss concealment for voice communications over IP networks," *IEEE Trans. Multimedia*, to appear in Dec. 2003.

[8] W. Verhelst and M. Roelands, "An overlap-add technique based on waveform similarity (WSOLA) for high quality time-scale modification of speech," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*, vol. 2, (Minneapolis, USA), pp. 554–557, Apr. 1993.

[9] S. B. Moon, J. Kurose, and D. Towsley, "Packet audio playout delay adjustment: Performance bounds and algorithms," *ACM/Springer Multimedia Systems*, vol. 5, pp. 17–28, Jan. 1998.

[10] J. Pinto and K. J. Christensen, "An algorithm for playout of packet voice based on adaptive adjustment of talkspurt silence periods," in *Proc. IEEE Conf. Local Comp. Networks*, (Lowell, MA), pp. 224–231, Oct. 1999.

[11] P. Agrawal, J.-C. Chen, and C. J. Sreenan, "Use of statistical methods to reduce delays for media playback buffering," in *Proc. IEEE Int. Conf. Multimedia Comput. Syst.*, (Austin, TX), pp. 259–263, June 1998.

[12] P. DeLeon and C. Sreenan, "An adaptive predictor for media playout buffering," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*, vol. 6, (Phoenix, AZ), pp. 3097–3100, Mar. 1999.

[13] S. Haykin, *Adaptive Filter Theory*. Upper Saddle River, NJ: Prentice Hall, third ed., 1996.

[14] D. Minoli and E. Minoli, *Delivering Voice over IP Networks*. New York, NY: John Wiley & Sons, Inc., 1998.

[15] J. Postel, "Internet protocol." RFC 791, Internet Engineering Task Force, Sept. 1981.

[16] A. S. Tanenbaum, *Computer Networks*. Upper Saddle River, NJ: Prentice Hall, third ed., 1996.

[17] J. Postel, "Transmission control protocol." RFC 793, Internet Engineering Task Force, Sept. 1981.

[18] J. Postel, "User datagram protocol." RFC 768, Internet Engineering Task Force, Aug. 1980.

[19] P. J. Smith, "Voice conferencing over IP networks," Master's thesis, McGill University, Montreal, Canada, Jan. 2002.

[20] D. O'Shaughnessy, *Speech Communications: Human and Machine*. New York, NY: IEEE Press, second ed., 2000.

[21] W. C. Hardy, *VoIP Service Quality: Measuring and Evaluation Packet-Switched Voice*. New York, NY: McGraw-Hill, 2003.

[22] B. Douskalis, *IP Telephony: The Integration of Robust VoIP Services*. Upper Saddle River, NJ: Prentice Hall, 2000.

[23] E. Mahfuz, "Packet loss concealment for voice transmission over IP networks," Master's thesis, McGill University, Montreal, Canada, Sept. 2001.

[24] J.-C. Bolot and A. Vega-García, "Control mechanisms for packet audio in the Internet," in *Proc. IEEE Infocom Conf. Comp. Commun.*, vol. 1, (San Francisco, CA), pp. 232–239, Mar. 1996.

[25] J.-C. Bolot, S. Fosse-Paris, and D. Towsley, "Adaptive FEC-based error control for Internet telephony," in *Proc. IEEE Infocom Conf. Comp. Commun.*, vol. 3, (New York, NY), pp. 1453–1460, Mar. 1999.

[26] J. Rosenberg, L. Qiu, and H. Schulzrinne, "Integrating packet FEC into adaptive voice playout buffer algorithms on the Internet," in *Proc. IEEE Infocom Conf. Comp. Commun.*, vol. 3, (Tel Aviv, Israel), pp. 1705–1714, Mar. 2000.

[27] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A transport protocol for real-time applications." RFC 1889, Internet Engineering Task Force, Jan. 1996.

[28] ITU-T Recommendation H.323, "Packet-based multimedia communication systems (prepublication version)," Nov. 2000.

[29] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg, "SIP: Session initiation protocol." RFC 2543, Internet Engineering Task Force, Mar. 1999.

[30] F. Cuervo, N. Greene, A. Rayhan, C. Huitema, B. Rosen, and J. Segers, "Megaco protocol version 1.0." RFC 3015, Internet Engineering Task Force, Nov. 2000.

[31] H. Schulzrinne, "RTP profile for audio and video conferences with minimal control." RFC 1890, Internet Engineering Task Force, Jan. 1996.

[32] D. Collins, *Carrier Grade Voice Over IP*. New York, NY: McGraw-Hill, second ed., 2003.

[33] S. Casner and V. Jacobson, "Compressing IP/UDP/RTP headers for low-speed serial links." RFC 2508, Internet Engineering Task Force, Feb. 1999.

[34] ITU-T Recommendation H.245, "Call signalling protocols and media stream packetization for packet-based multimedia communication system (prepublication version)," Nov. 2000.

[35] ITU-T Recommendation H.245, "Control protocol for multimedia communication (prepublication version)," Feb. 2000.

[36] F. P. Zhang, O. W. W. Yang, and B. Cheng, "Performance evaluation of jitter management algorithms," in *Proc. IEEE Canadian Conf. Elec. Comp. Eng.*, vol. 2, (Toronto, ON), pp. 1011–1016, May 2001.

[37] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, "RSVP: A new resource ReSerVation Protocol," *IEEE Network*, vol. 7, pp. 8–18, Sept. 1993.

[38] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An architecture for differentiated services." RFC 2475, Internet Engineering Task Force, Dec. 1998.

[39] W. A. Montgomery, "Techniques for packet voice synchronization," *IEEE J. Select. Areas Commun.*, vol. SAC-1, pp. 1022–1028, Dec. 1983.

[40] Y. J. Liang, N. Färber, and B. Girod, "Adaptive playout scheduling using time-scale modification in packet voice communications," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*, vol. 3, (Salt Lake City, UT), pp. 1445–1448, May 2001.

[41] V. Jacobson, "Congestion avoidance and control," in *Proc. ACM SIGCOMM*, vol. 18, (Stanford, CA), pp. 314–329, Aug. 1988.

[42] D. L. Mills, "Internet delay experiments." RFC 889, Internet Engineering Task Force, Dec. 1983.

[43] A. Kansar and A. Karandikar, "Jitter-free audio playout over best effort packet networks," in *ATM Forum Int. Symp.*, (New Delhi, India), Aug. 2001.

[44] A. Kansal and A. Karandikar, "Adaptive delay estimation for low jitter audio over Internet," in *Proc. IEEE Global Telecommun. Conf.*, vol. 4, (San Antonio, TX), pp. 2591–2595, Nov. 2001.

[45] C. J. Sreenan, J.-C. Chen, P. Agrawal, and B. Narendran, "Delay reduction techniques for playout buffering," *IEEE Trans. Multimedia*, vol. 2, pp. 88–100, June 2000.

[46] K. Fujimoto, S. Ata, and M. Murata, "Playout control for streaming applications by statistical delay analysis," in *Proc. IEEE Int. Conf. Commun.*, vol. 8, (Helsinki, Finland), pp. 2337–2342, June 2001.

[47] K. Fujimoto, S. Ata, and M. Murata, "Statistical analysis of packet delays in the Internet and its application to playout control for streaming applications," *IEICE Trans. Commun.*, vol. E84-B, pp. 1504–1512, June 2001.

[48] K. Fujimoto, S. Ata, and M. Murata, "Adaptive playout buffer algorithm for enhancing perceived quality of streaming applications," in *Proc. IEEE Global Telecommun. Conf.*, vol. 3, (Taipei, Taiwan), pp. 2451–2457, Nov. 2002.

[49] J.-C. Bolot, "End-to-end packet delay and loss behaviour in the Internet," in *Proc. ACM SIGCOMM*, (San Francisco, CA), pp. 289–298, Sept. 1993.

[50] A. Shallwani and P. Kabal, "An adaptive playout algorithm with delay spike detection for real-time VoIP," in *Proc. IEEE Canadian Conf. Elec. Comp. Eng.*, (Montreal, Canada), May 2003.

[51] V. Paxson, G. Almes, J. Mahdavi, and M. Mathis, "Framework for IP performance metrics." RFC 2330, Internet Engineering Task Force, May 1998.

[52] S. B. Moon, *Measurement and Analysis of End-To-End Delay and Loss in the Internet*. PhD thesis, University of Massachusetts Amherst, Amherst, MA, Feb. 2000.

[53] D. L. Mills, "Improved algorithms for synchronizing computer network clocks," *IEEE/ACM Trans. Networking*, vol. 3, pp. 245–254, June 1995.

[54] D. L. Mills, "Network time protocol." RFC 1305, Internet Engineering Task Force, Mar. 1992.

[55] D. L. Mills, "Internet time synchronization: The network time protocol," *IEEE Trans. Communications*, vol. 39, pp. 1482–1493, Oct. 1991.

[56] K. C. Claffy, G. C. Polyzos, and H.-W. Braun, "Measurement considerations for assessing unidirectional latencies," *J. Internetworking: Res. and Experience*, vol. 4, pp. 121–132, Sept. 1993.

[57] D. L. Mills, "On the accuracy and stability of clocks synchronized by the network time protocol in the Internet system," *ACM Comp. Commun. Review*, vol. 20, pp. 65–75, Jan. 1990.

[58] V. Paxson, "On calibrating measurements of packet transit times," in *Proc. ACM SIGMETRICS*, (Madison, WI), pp. 11–21, June 1998.

[59] W. Lewandowski, J. Azoubib, and W. J. Klepczynski, "GPS: Primary tool for time transfer," *Proc. IEEE*, vol. 87, pp. 163–172, Jan. 1999.

[60] G. Almes, S. Kalidindi, and M. Zekauskas, "A one-way delay metric for IPPM." RFC 2679, Internet Engineering Task Force, Sept. 1999.

[61] M. A. Lombardi, L. M. Nelson, A. N. Novick, and V. S. Zhang, "Time and frequency measurements using the Global Positioning System," *Cal Lab Int. J. Metrology*, pp. 26–33, July–Sept 2001.

[62] W. Lewandowski and C. Thomas, "GPS time transfer," *Proc. IEEE*, vol. 79, pp. 991–1000, July 1991.

[63] S. B. Moon, P. Skelly, and D. Towsley, "Estimation and removal of clock skew from network delay measurements," in *Proc. IEEE Infocom Conf. Comp. Commun.*, (New York, NY), pp. 227–234, Mar. 1999.

[64] H. Melvin and L. Murphy, "Time synchronization for VoIP quality of service," *IEEE Internet Comput.*, vol. 6, pp. 57–63, May-Jun 2002.

[65] V. Paxson, "End-to-end routing behavior in the Internet," *IEEE/ACM Trans. Networking*, vol. 5, pp. 601–615, Oct. 1997.

[66] J. Postel, "Internet control message protocol." RFC 792, Internet Engineering Task Force, Sept. 1981.