INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI®

Université d'Ottawa · University of Ottawa

Packet Voice over non-QoS Networks

by

© **Tie Xu**

A thesis submitted to the
School of Graduate and Research
In partial fulfillment of the requirements for the degree of

Master of Applied Science
in Electrical Engineering

Ottawa-Carleton Institute of Electrical and Computer Engineering
School of Information Technology and Engineering

Faculty of Engineering
University of Ottawa

Ottawa, Ontario, 1999

Abstract

Past communication networks were characterized by separate networks for data and voice. The integration of data and voice in a unique network rapidly gained more and more interest. This integration can provide bandwidth efficiency and new integrated services. The traditional packet switched network cannot be directly applied to voice communication, because it doesn't provide quality of service (QoS). One solution is to modify the network structure to provide new real-time services. Application adaptation provides an alternative solution, with which we can build real-time applications upon currently available network services. The adaptations include bandwidth control, lost packet recovery, delay jitter control, and operating scheduling compensation. The key problem is delay jitter compensation. In this thesis, the application adaptation solution is discussed and algorithms are surveyed. A simulation tool is developed to study the jitter management problem. By managing the tradeoff between display latency and gap frequency, and by dynamically adjusting the display latency according to network conditions, we can provide acceptable voice transmission as long as the average bandwidth is sufficient.

Acknowledgements

I would like to express my deepest appreciation to my supervisor, Prof. Nicolas Georganas, for his guidance, encouragement and financial assistance throughout my graduate studies.

I would also like to express my thanks to Nancy Poon, Don Wade, Eber Mello, and Craig Frish for their useful comments and for the fruitful discussions we had during the one year intern project in MITEL. I would like to give special thanks to Ebber Mello for his help in building the simulation tool.

Table of Contents

Chapter 1 Introduction

1.1 Voice Data Integration

Traditional voice service is carried by the Public Switch Telephone Network (PSTN). PSTN is a circuit switched network characterized by guaranteed bandwidth and delivery. In such a system, when a call arrives, a transparent line is established for realizing smooth conversation between the origin and destination terminals. A fixed channel capacity is assigned to this connection throughout the call period. The delay is small and constant, so the quality of voice transmission can be guaranteed. But at the same time a lot of bandwidth is wasted, because the channel capacity must be assigned to the call according to its peak data rate, not the average data rate, and the line must be held even if no talk-spurt is transmitted.

In contrast, a totally separate network system, the packet switched network, was designed to carry data communications. In a packet switched network, data are grouped as packets, and they are transmitted in a store and forward mode. The different transmissions share the same media in a statistical way. A packet switched network is characterized by its bandwidth efficiency (shared bandwidth). But because of the statistical nature of packet switching, each packet can encounter a different amount of delay in traversing the packet network. And in a routed network, because each packet could be routed independently, packets may even arrive at the destination out of order. The other problem of packet switched networks is packet loss. In a store and forward network, packets are queued in switches, bridges, and routers along the path to the destination. Because of possible

congestion of the network and the limit length of the queue, the packets could be discarded. The delay variance is not a problem to traditional data communication. The out of order packet and packet loss can be recovered by upper layer network protocols, such as TCP. But these features make packet switch networks unsuitable for carrying voice service. Voice traffic, with its real-time characteristic, is more subtle to variance of delays and discontinuity than data traffic. The upper layer protocols, that can recover out-of-order packets and packet loss, may introduce rather long delay. Thus they can't be applied to voice traffic.

In recent years, there was increasing interest of data and voice integration in a single network. The first motivation of integration is sharing network resources among data and voice. A lot of efforts have been spent in accommodating voice services through packet switched networks. Another motivation is the ability of providing voice and data integrated services, like multimedia conferencing. This is considered, in the long run, as the major benefit from data voice integration. Other benefits of voice data integration include equipment commonality, combined network operations, maintenance, and administrative policies.

The fast developing of Internet is another force pushing data and voice integration. Internet provides a unique network to bind different types of datalink layer packet switched networks all over the world. It has the most connectivity and availability. In recent years, inexpensive hardware for processing digitized audio is rapidly becoming available for desktop personal computers. At the same time, high network bandwidth and high performance desktop computers at relatively low price become widely available. A virtual network Multicast Backbone (MBONE) has been built upon the Internet to

provide a multicast service, which make real-time media broadcast possible. It is attractive in providing voice related applications over the Internet. There have been many efforts in developing multimedia applications. These efforts resulted in some application prototypes and even a few commercial products, such as multimedia conferencing tools and Internet phone. But there is still a long way to providing real valuable real-time services over the Internet.

To accommodate voice communication over a packet switched network, we must first digitize the speech at a uniform rate, and then organize these data into constant length packets with packet headers just as other data packets. The packets are transmitted through the store-and-forward network and arrive at the receiver network node, where the reverse process is applied to the speech packets and the speech is played out. There are several technologies involved in this process. They are speech coding, packet switching, real-time support, and desktop applications. The key problem is real-time support. As we know, voice communication requires guaranteed bandwidth and short response time, that can not be provided by traditional packet switched networks.

One solution is to modify the packet network structure to make it provide new services to support real-time traffic. The new services must provide QoS to traffic, like guaranteed bandwidth, minimal delay, etc. These efforts will be tremendous and will touch all the layers of a network, from the datalink layer, network layer, transport layer, to the application layer. An alternative solution is application adaptation, which means making the application adaptive to available network services and providing acceptable real-time applications. Compared to the first solution, the application adaptation is limited. But it also has some benefits. The first benefit is that it makes use of the current network

structure and available network services. Thus it is much more applicable than the first solution. Given sufficient bandwidth, it can provide pretty good real-time applications. The second benefit is that, in the long run, the application adaptation still has its value. New network real-time services can not be applied to the whole network at a jump. There will be a long coexistence of traditional and upgraded networks. Application adaptation helps to provide real-time services in such a heterogeneous network environment.

1.2 Organization of This Thesis

The next chapter surveys packet voice technologies. These technologies include speech coding algorithms, LAN technology, public packet switched technology, new network services, application adaptation technology and related protocols, voice over IP applications and related protocols.

Chapter 3 investigates different aspects of application adaptation, with focus on delay jitter management. In bandwidth control, a simple control strategy is introduced. In packet loss recovery, two kinds of strategies are introduced. The network delay jitter problem and related synchronization problems are discussed. A series of delay jitter management algorithms is described.

Chapter 4 is a simulation study on delay jitter management. A platform is developed to implement and evaluate different jitter management algorithms. An end-to-end delay model is applied to add delay to the tool. A series of experiments are performed to evaluate the feature of the jitter compensation algorithms.

Conclusion is given in chapter 5.

1.3 Thesis Contributions

The main contributions of this thesis are the following:

We have investigated packet voice performance over a non QoS network, and made a complete survey on application adaptation algorithms. These algorithms include bandwidth control, packet loss recovery, delay jitter control, and operation system scheduling compensation algorithms.

We have developed a simulation system for studying jitter control algorithms. This system uses an audio file as input, and can generate output as an audio file. A network end-to-end model is implemented to generate delay jitter, and the system can take other sources of delay jitter, such as delay generated by other models or delay recorded from a real network. We have implemented a series of jitter management algorithms in this system, and evaluated the performance of these algorithms. We modified the dynamic display latency adjustment strategy, by setting a range to the change of silence duration. We have implemented an adaptive silence detection algorithm in this system. We also introduced a hangover strategy into this algorithm to merge small silence periods.

Chapter 2 Overview of Packet Voice Technology

2.1 Audio Digitization and Codec

In a packet voice system, audio must be transformed into digital signal in order to be processed and transferred. In most cases, digital audio compression is also required when network bandwidth is limited or costly.

2.1.1 Audio Sampling and Digitizing

In audio digitizing, an audio wave is measured at regular intervals. The sampling rate is decided by the frequency range of the audio. Because audio with bandwidth between 40 Hz and 4 kHz can effectively represent human voice can, according to Nyquist's theory, in order to record human voice, the sampling rate should be at least 8 kHz. To match human hearing, which has a frequency range between 20 Hz and 20 kHz, 40 kHz of sampling rate is necessary.

Each sample is transformed into digits. Usually, we use 8 bits to 16 bits to represent a sample. With 8 bits digitization, we have 48 dB audio, and with 16 bits we have 96 dB. Analog voice can be digitized linearly or non-linearly. Uniform Pulse Code Modulation (PCM) is a linear encoding. A-law and μ-law PCM are non-linear encodings, which use a kind of logarithmically spaced quantifier. Non-linear encoding can give more accuracy to low energy parts of voice signal.

2.1.2 Audio Encoding and Compression

An uncompressed PCM audio with 8 kHz sampling rate and 8 bits quantification will consume 64 kbps bandwidth. Although a typical 10M Ethernet LAN can handle it easily, it is unacceptable for multimedia conference applications and wide area networks applications. The principle parameters of an audio codec are speech quality, bit rate, complexity and delay(frame size). Optimizing for one or more of these parameters usually means sacrificing performance in other areas. A tradeoff must be made in designing or choosing a codec.

The simplest audio codec is logarithmic PCM (μ-law and A-law). It allows a larger range of values to be covered with the same number of bits for a given speech quality. μ-law and A-law, which are commonly used transformations, use 8 bits to represent the same range of values that would be represented by uniform PCM in 14 bits. The compression rate is 1.75:1. The μ-law and A-law PCM encoding methods are defined in ITU-T recommendation G.711, which is under the H.323 umbrella.

Adaptive Differential Pulse Code Modulation (ADPCM) is an enhancement of PCM. It makes use of the similarity of adjacent samples. Instead of encoding the samples, it encodes the difference between adjacent samples and its predicted value. The prediction and quantification parameters are adaptive to the signal characteristics. Generally, the compression ratio is 2:1 compared to A-law or μ-law PCM. The ITU-T has defined several ADPCM methods (G.721, G.722, G.723, G.726, G.727).

Linear Predictive Coding (LPC) is a more powerful encoding method compared to the above encodings. But it is designed specially for speech and it is more complex. LPC starts with the assumption that the speech signal is produced by a buzzer at the end of a

tube. The vocal tract forms the tube, which is characterized by its resonance, which are called formants. LPC analyzes the speech signal by estimating the formants, removing their effects from the speech signal, and estimating the intensity and frequency of the remaining parts. Federal Standard 1015 defines a simple LPC. It provides intelligible speech transmission at 2400 bps. Code Excited LPC (CELP) is a more complex LPC. It analyzes the residue, compares it to all the entries in a codebook, chooses the entry, which is the closest match, and just sends the code for the entry. ITU-T G.728 uses a variation of CELP. G.728 is also specified by H.320 and H.323.

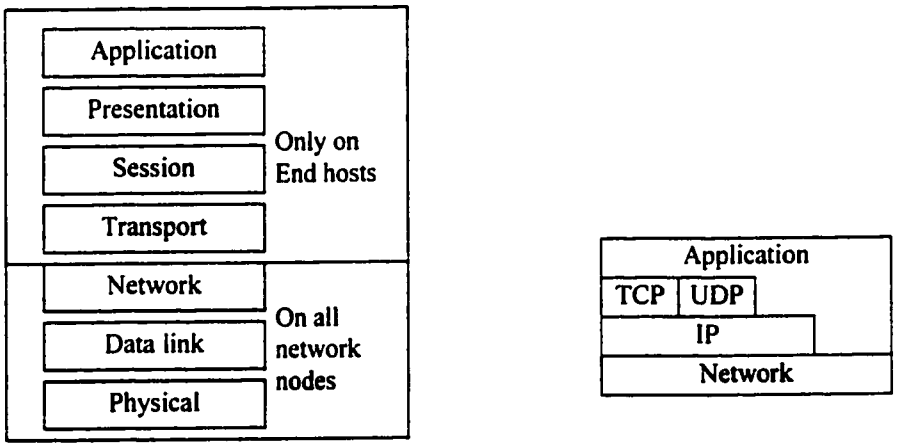
2.2 Packet Switched Network

2.2.1 Structure of Network

In a packet voice system, voice packets are carried by the packet switched network. In a complex network system, we often use a layering strategy. This provides two benefits. First, it decomposes the problem of building a network into more manageable components. Instead of dealing with a complex problem, we deal with a layer at a time and each layer solves one part of the problem. Second, it provides a more modular design. Each layer offers functions to its upper layer. It is easy to add some functions by making use of the functions offered by its lower layer without changing other functions in its own layer.

The ISO's Open Systems Interconnection (OSI) defines a seven-layer model (Figure 2.1). From the bottom, the physical layer handles the transmission of raw bits over a communication link. The data link layer collects a stream of bits into a larger aggregate, called a frame. Network adapters typically implement the data link level, meaning that

frames, not raw bits, are actually delivered to hosts. The network layer handles routing among nodes within a packet switched network. At this layer, the data unit is called packet. The lower three layers are implemented on all network nodes. The transport layer implements a process-to-process channel. It offers some services to the upper layers, such as connection channels and connectionless channels. There are not strict definitions for the top three layers. The transport layer and higher layers typically run only on the end hosts and not on the intermediates switches or routers.



a. OSI protocol structure

b. Internet protocol structure

Figure 2.1 OSI protocol structure and Internet protocol structure

In the Internet domain, the layered structure is not so complicated. In the Internet we use a four-layer model. This is shown in Figure 2.1b. The lowest level is network layer. It covers the function of data-link layer and physical layer of OSI model. The protocols in this layer are implemented by a combination of hardware and software. Examples are Ethernet and FDDI. The second layer is Internet Protocol (IP). This protocol supports the interconnection of different types of data-link layer network into a single network

domain. The third layer has two protocols TCP and UDP. TCP and UDP provide virtual channels to the application layer. TCP provides a reliable channel and UDP provides an unreliable channel. Both TCP and UDP are end side only. They are only implemented in end nodes.

2.2.2 Datalink Layer Network Protocols

There are several protocols located in this layer. Basically, they can be divided into two categories. One is local area media access protocols like Ethernet, Token Ring, and FDDI. The other is wide area switched-network protocols, like X.25, Frame Relay, ATM.

Ethernet (CSMA/CD)

Ethernet is the most successful and most widely used local area networking technology. It was developed from an early packet-radio network, called ALOHA. In Ethernet, all stations share the same medium. A carrier sense strategy is applied to solve the medium access problem. Each station senses the medium and sends packets only when the medium is free. When collision is detected, the transmission is canceled and the retransmission is applied with a random backoff. The typical Ethernet is 10 Mbits/sec, defined by IEEE802.3. The most important feature of Ethernet is that it works well under light traffic and its performance drops sharply when the network traffic reaches a threshold.

Token Ring & FDDI

Token ring is another important LAN technology. There are several implementations of token ring, such as IBM's 4 Mbps, IEEE 802.5 token ring, and FDDI. It is also a shared media network just as Ethernet. But its media access strategy is quite different. In a token ring network, all stations are connected in a ring. Data flows in a particular direction around the ring. A token, which is a special frame, circulates around the ring. If a station wants to send a frame, it must wait for its turn to get the free token. It attaches its frame to the token frame, marks it occupied, and sends it. When the frame arrives at the destination, the token is free (in delayed free mode). FDDI makes an extension to the token ring. It has two independent optical fiber rings that transmit data in opposite directions. The second ring is not working in normal operation. But when the normal ring is broken, the ring loops back on the second ring to form a complete ring. The media access algorithm of FDDI is also more complicated than legacy token ring. It introduces a concept of token holding time (THT) and target token holding time (TTHT), which gives synchronized data high propriety. This feature makes FDDI more suitable to transfer time related data (voice and video), compared to other LAN techniques. But because of its complexity and high cost, it is not widely applied.

Public packet switched network (X.25, Frame relay)

X.25 provides users with WAN connectivity across public data networks. It defines a network model with data terminal equipment (DTE) and data circuit-terminating equipment (DCE). DTEs connect to DCEs, which connect to packet switching exchanges and other DCEs within a public switched network (PSN)and, ultimately, to other DTEs.

X.25 specifies a 3 layer model which can be mapped to layers 1 through 3 of the OSI reference model. Layer 3 X.25 describes packet formats and packet exchange procedures. Layer 2 X.25 defines packet framing for the DTE/DCE link. Link 1 X.25 defines the electrical and mechanical procedures for a physical medium. The end-to-end communication between DTEs is accomplished through a bidirectional association called a virtual circuit. Virtual circuits can either be permanent or switched(temporary). Permanent virtual circuits are commonly called PVCs; switched virtual circuits are commonly called SVCs. PVCs are typically used for the most-often-used data transfers, while SVCs are used for sporadic data transfers. Layer 3 X.25 is concerned with end-to-end communication involving both PVCs and SVCs.

Frame Relay was originally conceived as a protocol for use over Integrated Services Digital Network (ISDN) interfaces. But now, like X.25, frame relay provides a packet-switching data communications capability that is used across the interface between user devices and network equipment. However, Frame Relay differs significantly from X.25 in its functionality and format.

As an interface between user and network equipment, Frame Relay provides statistically multiplexing (a media sharing mechanism that gives each node equal chance but can give more bandwidth to nodes with more data to transfer) over a single physical transmission link. Compared with systems that use only time-division-multiplexing (TDM) techniques, Frame Relay provides more flexible and efficient use of available bandwidth. Another important characteristic of Frame Relay is that it exploits the recent advances in WAN transmission technology. Unlike X.25 that was developed when analog transmission systems and copper media were predominant, Frame Relay is designed upon much more

reliable fiber media /digital transmission. It includes a cyclic redundancy check (CRC) for detecting corrupted bits, but it doesn't include any protocol mechanisms for correcting bad data. It leaves these to be performed at higher protocol layer. The other feature of Frame Relay is it does not have explicit, per-virtual-circuit flow control, because many upper-layer protocols have this functionality. Frame Relay can be used as an interface to either a public available carrier-provided service or to a network of privately owned equipment.

Cell switched network (ATM)

Asynchronous transfer mode (ATM) technology is based on the efforts of the ITU-T study group XVIII to develop Broadband Integrated Services Digital Network (BISDN) for the high-speed transfer of voice, video, and data through public network.

ATM has become a tremendously important network technology in recent years. It is a connection oriented packet-switched network technology. It has a fixed size packet, called cell. Each cell consists of 5 octets of header information and 48 octets of payload data. ATM is a cell-switching and multiplexing technology that combines the benefits of circuit switching (constant transmission delay and guaranteed capacity) with those of packet switching (flexibility and efficiency for intermittent traffic). ATM defines the network-user interface (referred to as UNI) and network-node interface (referred to as NNI).

ATM is an asynchronous mechanism. It differs from synchronous transfer mode in that, ATM time slots are made available on demand, with information identifying the source of the transmission contained in the header of each ATM cell. With TDM, if a station has nothing to transmit when its time slot comes up, that time slot is wasted. While if a

station has a lot of information to send, it must wait for its turn, although there are a lot of empty slots. With ATM, a station can send cells whenever necessary.

ATM defines protocol layers. They are the physical layer, ATM layer, ATM adaptation layer and higher layers. The ATM layer and the ATM adaptation layer are roughly analogous parts of the data link layer of the OSI model, and the ATM physical layer is analogous to the physical layer of the OSI model. ATM has its unique feature that it defines different adaptation layers that represent different level of quality of service (QoS). AAL1 provides constant bit rate, which is appropriate for transporting telephone traffic and uncompressed video traffic. AAL3/4 provides variable bit service, which was designed for a network service provider and is closely aligned with Switched Multimegabit Data Service (SMDS). AAL5 provides available service, which is used to transfer most non-SMDS data, such as classical IP over ATM and LAN emulation.

A signaling protocol is conducted between end-device and ATM switches. An end device uses the signaling protocol to build a connection and request for its QoS. The request is sent to its directly connected switch. The switch propagates the request along the path to the endpoint. If any switch cannot accommodate the requested QoS parameters, the request is rejected. Otherwise, an accept message is issued by the receiver node, and it is propagated back to the originator of the request. The switches along the path set up a virtual circuit.

2.2.3 Internet

Internet (TCP/IP) is a set of protocols that combine different physical networks to form a single logical network and offers some end-to-end services.

An IP service model can be thought of as having two parts: An addressing scheme and packet forward & delivery. IP uses a global addressing scheme. An IP address is made up of two parts: network address and host address. The network part identifies the network to which the host is attached. The host part identifies each host uniquely on that network. An IP packet is composed of a header and data. The header carries all the information needed to deliver the packet. But the IP delivery is "best effort", which means there is no guarantee of delivery. This is also called unreliable service. It is possible to build some reliable services upon this unreliable IP service by putting some extra functionality.

UDP and TCP are two basic end-to-end services built on top of IP. UDP is a simple extension to IP service. It only adds a demultiplexing service to let multiple application processes on each host to share the same IP address. Because of its light overhead, real-time services are always built on UDP. We will revisit this topic in later sections.

TCP is a more sophisticated transport protocol. It offers a connection-oriented service and it also guarantees reliable in-order transmission stream. It also has a sliding window flow-control scheme that can make the transmission adaptive to the network traffic. But compared with UDP, TCP introduces a lot of overhead such as retransmission of lost packets. So it is hard to build real-time services upon TCP.

Because of the limitation of current IP and also because of the scaling problems of address space, a next generation IP, called IPv6, is being developed and defined by standard groups. First, IPv6 provides a 128-bit address space, which can carry $3.4*10^{38}$ nodes. IPv6 also brings some other changes such as support for real-time service, security service, etc. We will discuss more about real-time services in later sections.

2.2.4 Multicast Service and Mbone

Multicast is critical to many multimedia transmission applications. It is supported by most LANs such as Ethernet and Token Ring. But in a large scale routing network, multicast is not easy to implement. A multicast address is defined as class D address in IP address space, in the range of 224.0.0.0 to 239.255.255.255. A multicast address is a logical address (not a physical address). It is the identifier of a host group. Senders use this group address as destination address to send IP datagrams, and all the group members can receive the data. Hosts that wish to receive the data can join the group by sending IGMP messages.

Multicast routing is a very complicated problem. There are several different routing protocols that multicast routers can use to deliver multicast packets, including DVMRP (Distance Vector Multicast Routing Protocol), MOSPF (Multicast Open Shortest Path First) and PIM (Protocol Independent Multicast). But so far only a part of Internet routers support multicast routing. A virtual multicast network, called Mbone, is built upon the current Internet. Multicast packets are tunneled when transferring through unicast routers.

2.3 Voice over Packet Switched Network

2.3.1 Overview and Motivation

The transmission of voice over packet switched networks has been an active research area since the late 70's. The investigations were on both LAN technologies and ARPANET, which is the forerunner of Internet.

The first motivation of these efforts is efficient utilization of channel capacity. Current communication systems are characterized by separate networks for voice and data. With

the increasing bandwidth of packet data network and developing of high speed terminals, it is considered that transmitting voice and data in a single network is more efficient. The existing workstations as audio terminals and existing LAN/WAN infrastructure may be used to supplement or replace the traditional centralized telephone exchanges system.

The second motivation is that the integration of voice and data gives some new features to voice communication and even brings some new applications that can not be carried out by traditional telephone services or by traditional data network services separately. For example, with voice data integration, there could be a tele-learning application, in which each recipient can share the voice of lecture and also the data (text or graph) that is related to the lecture.

In the LAN area, Ethernet was one of the earliest to be implemented and then developed to be the most popular architecture. There has been interest on the use of Ethernet for voice and other real-time application since the early years of Ethernet. In [33], experiments show that a 3Mbps Ethernet (experimental Ethernet) was capable of supporting about 40 simultaneous 64-Kbps two-way voice conversations with acceptable quality. In [13], there was a more complete investigation on the Ethernet capacity for voice communication. The simulation study shows that, in the absence of data, an Ethernet can support 60 pairs of conversations with almost zero delay. But after that point, when the number of conversations grows, the average delay increases sharply and the standard deviation of delay increases faster than the average value. The simulation also shows that, with data traffic introduced as background traffic, the performance of voice transmission decreases very much. In this case, with the number of conversations

increasing, there is no longer any zero delay region, and both the average delay and the standard deviation increase much faster.

Instead of using an existing LAN framework, some new LAN protocols [29] [30] were proposed to support real time traffic. In [29], an Ethernet compatible protocol, multi-class real time (MCRT) protocol was proposed. In this protocol, channels are shared between MCRT frames and normal data frames, and real time service is guaranteed by an MCRT chain. In [30], a reservation-based Carrier Sensitive Media Access (CSMA) protocol was presented to let LAN carry real time data. It extends the CSMA/CD to include a bandwidth reservation policy that accommodates the demands of real-time applications with periodic sources. Under this protocol, when a data frame collides with a periodic source, the data frame backs off, but the periodic source obtains the media. If two periodic sources collide, one of them shifts to the next available slot.

In the Internet area, voice over IP has been an interesting topic since the beginning of ARPANET. Traditional Internet does not provide guaranteed resources such as bandwidth or guaranteed performance measures such as maximum delay or maximum packet loss rate. The only available end-to-end services on Internet are TCP and UDP. Many voice communication experiments and investigations are based on UDP/IP. There are two approaches to solve this real-time transmission problem. One solution is to extend current protocols and switch scheduling disciplines to provide Quality of Service. This approach requires that admission control, policing, reservation, and sophisticated scheduling mechanisms be implemented in the network. RSVP (Reservation Protocol) is such a protocol. The design, analysis, and evaluation of such mechanisms is an active research area.

Another approach is using application adaptation based on existing best-effort network services. The adaptation includes bandwidth control, packet loss recovery, delay jitter compensation. Because the adaptation is only applied at end points and doesn't require any change to the network architecture, it is much easier to implement than the first approach. There are also many interests in this area and there are many applications based on adaptation strategy.

There are some other issues in packet voice communication, such as voice compression, admission control, voice conference control that are not covered by this work.

2.3.2 New Network Services

As we mentioned before, in order to support real time communication in data network, some new network infrastructures and new network services come out. RSVP(resource reservation protocol) is such a protocol.

RSVP introduction

RSVP is a resource reservation protocol designed for an integrated services Internet. RSVP is applied both in end hosts and intermediate routers. Hosts use RSVP to request a specific QoS (quality of service) from the network. RSVP is also used by routers to make sure all the routers along the path of a data stream establish and maintain the required service.

RSVP is receiver-oriented. That is to say the receiver is responsible for initiating and maintaining the resource reservation. RSVP is not a routing protocol. Instead, it relies on current and future routing protocols. RSVP is designed to support both unicast and

multicast IP data flow. It is designed to scale well for very large multicast groups. Although RSVP is a protocol to support QoS across an IP network, it only defines how to build and maintain a certain real-time service in the Internet and it doesn't define the detail contents of QoS. The QoS is determined by the integrated service models and is generally opaque to RSVP.

RSVP scenario

Figure 2.2 shows how RSVP works in hosts and routers. In an end host, an application makes reservation request to a RSVP daemon, which is running in local host. The request is then passed to two local decision modules, "admission control" and "policy control". Admission control determines whether the node has enough resources to fulfill the QoS request, and the policy control determines whether the application has the permission to make the request. If the request passes both modules, the QoS parameters will be set into the packet classifier and packet scheduler. The classifier and scheduler implement QoS for each incoming packet. In a router, it is almost the same process, except that, first the reservation request is not from an application, but from a down stream hop, and second the RSVP daemon calculates a new QoS request and propagates it to an up stream hop.

RSVP reservation model

Each reservation request contains two parts, " flowspec" and "filterspec". The "flowspec" specifies QoS. The "filterspec" defines the set of data packets, on which the QoS will be applied. The "filterspec" will be used to set parameters in packet classifier. Then it is used to select a session or a subset of a given session. How to select a subset is up to the

application. For example, the application layer header may be defined to divide the session into different subsets and give different QoS to different subsets. The "flowspec" is used to set parameters in packet scheduler. If the link layer has its own QoS module, then the packet scheduler is responsible to map the QoS in "flowspec" into link layer QoS standard. The scheduler may also allocate other system resources, such as CPU time and buffers.

The reservation requests are sent by receivers and go through each router until they reach the senders. At each node, if the request passes the admission control and policy control, the QoS and filterspec are set into packet scheduler and classifier. Otherwise the request is rejected and an error message is sent. If the request succeeds, then it is propagated to upstream hop. The propagated request may not be the same with the received request for each node. The reasons are, first, the "flowspec"may very from hop to hop, second, in multicast, the router may merge the requests from different receivers and calculate a new "flowspec"for the upstream.

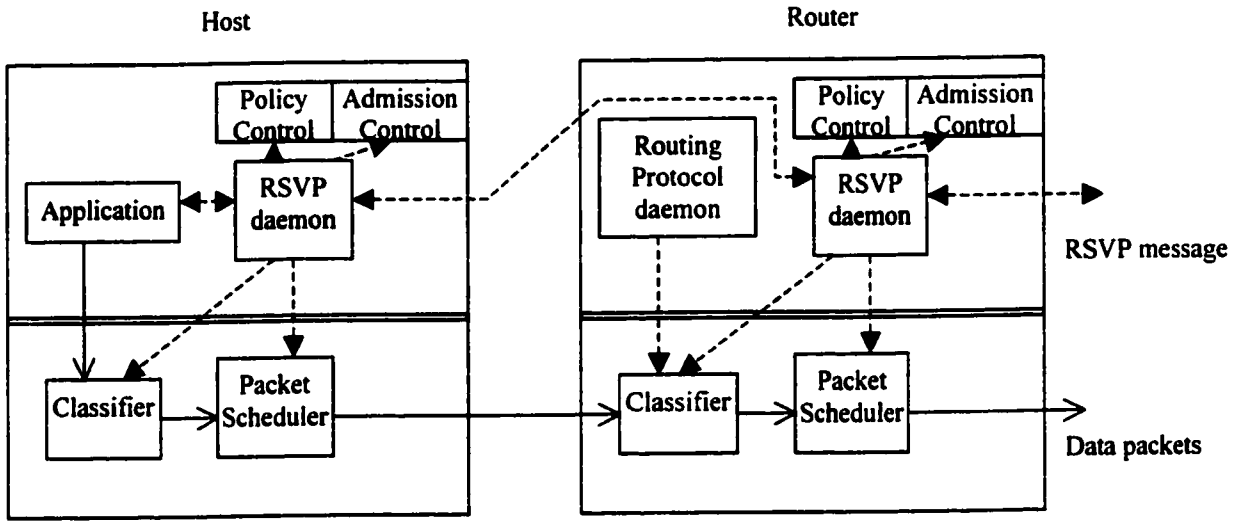


Figure 2.2 RSVP in host and router

RSVP mechanisms and messages

In fact, there are two steps in resource reservation. Before reservation requests are sent, RSVP sender has to send "path" message along the routes provided by routing protocols to the receiver. The path message records the "path state" of each node. The message will be used by the receiver to send the reservation request. A path message also contains some other important information: 1) Sender template, which describes the format of data packet. It could be used to set parameters in the packet filter. 2) Sender Tspec, which defines the traffic characteristics of a data stream. It could be used to prevent over-reservation. 3) Adspec, which contains information gathered from each node. The information can be used by the receiver to construct or dynamically adjust an appropriate reservation request.

During transmission, RSVP maintains a "soft state" in routers and hosts. The state is created and periodically refreshed by "path" and "reservation" message. It is deleted when no refresh messages arrive after a timeout or a "teardown" message is received. When the "soft state" is deleted, all the reserved resources are released. This mechanism is designed to accommodate dynamic path change and reservation change.

The third RSVP message is "tear down". It is sent by an end host when the application finishes or by routers when refresh timeout is reached. "Tear down" messages remove path and reservation state at a node immediately on arrival.

Applicability and coexistence with non-RSVP clouds

RSVP is on top of Internet and offers real-time service. But in order to implement RSVP, the link layer protocol must support QoS. The QoS may not be in the same format with

what is defined in integrated service Internet. In such case, a mapping between the two QoS specifications is necessary. For example, ATM is a QoS supported link layer protocol. If RSVP is run on top of IP over ATM, the RSVP scheduler should map a QoS request into an ATM service type.

Because it is impossible to deploy RSVP throughout the entire Internet at the same time, it is necessary to make RSVP coexistent with a non-RSVP cloud. RSVP is designed to operate correctly through such a cloud. Because RSVP uses a normal routing protocol just as other data packets, the "path" message will not be affected by non-RSVP routers. A "reservation" message will be forwarded directly to the next hop by a non-RSVP router without any change. But then, because the non-RSVP routers only offer best-effort service, the end user may not get the required service.

2.3.3 Application Adaptation

An alternative approach is to adapt applications to the service provided by the existing network. In the Internet domain, this means that we can build real-time transmission based on best effort services, TCP/IP or UDP/IP. Compared to the above approach, the adaptive approach has many benefits. First, the current network is a simple non-QoS environment. With adaptive strategy and appropriate bandwidth, we can offer acceptable real-time services. Second, because even in the future it is impossible to implement a reservation strategy (like RSVP) across the entire network, we still need application adaptation in a heterogeneous environment.

The network congestion is the key problem to overcome. It affects the transmission of packets in two aspects. First, in the local area network, all the stations share the same

media. When congestion occurs, a workstation encounters long and unanticipated delay when attempting to access the network. Second, in a wide area network, a packet passes a series of routers to arrive at the destination. When the congestion increases, the accumulated queuing delay in these routers can be long and unanticipated. And in both cases, because of the limited length of the queue, the packets can be lost. So, in order to give an acceptable real-time transmission, we must do load variation adaptation, jitter adaptation, and packet loss adaptation.

The three problems are related with each other. The common cause is network congestion. There are two types of congestion in network load. The congestion in a scale of millisecond, which is caused by short bursts of traffic on the network, is called short term congestion. A jitter control mechanism is designed to overcome this kind of congestion. A typical jitter control method is the buffering strategy. By setting an appropriate buffer depth at the receiver end, we can accommodate the variation of delay, and at the same time offer acceptable display latency.

But this jitter control mechanism doesn't work in a situation of long term congestion. A long-term congestion is caused by gross changes in network traffic that persist for hundreds of milliseconds. In long-term congestion, the throughput of frames between a transmitter and a receiver may not be sufficient to sustain normal display rates. This will result in long end-to-end latency, large jitter, and frequent and consecutive packet loss. In accommodating long-term congestion, the only way is adjusting the transmission rate. It is not so easy to change the audio transmission rate in a wide range. Audio codec is not like video, where bits stream can be easily controlled by dropping less important bits. One possible way is to use a panoply of audio codecs. For example, we can accommodate

the network traffic by switching among PCM (at 64 kb/s), various ADM(between 16 kb/s and 48 kb/s), GSM (at 11 kb/s), and LPC (below 5 kb/s) coders. A feedback mechanism also needs to be built between a sender and a receiver to provide the sender with current network traffic information.

Packet loss recovery is another method to improve the quality of audio transmission. There are two kinds of packet loss. As mentioned before, the network itself may lose packets because of traffic congestion. The other kind of packet loss is from those packets with too long delay. The nature of audio requires that packets should be played continuously. With jitter accommodation, the delay variation can be partly solved. But there are still some packets that arrive later than the threshold. These packets are also considered lost. The aim of packet loss recovery is to construct a suitable dummy packet at the receiver, so that the loss is as imperceptible as possible. Voice reconstruction techniques can be split into two categories; receiver only, and combined source and channel techniques. Receiver only techniques are those that try to reconstruct the missing packets of speech solely at the receiver, possibly from correctly received packets preceding that which was lost. Combined source and channel techniques are those that try to reconstruct the lost packets by either arranging for the transmitter to code the speech in such a way as to be robust to packet loss, or by transmitting extra information to help with reconstruction.

2.3.4 Real-Time Transport Protocol

To build real time services on top of the current Internet service, it is necessary to convey time-related information in packets. TCP or UDP do not have such a feature. Although it

is possible to solve this problem in the application layer, it is better to build a separate layer between the application layer and the network layer to offer a real-time service. This can give a standard common interface to all real-time applications and makes them be able to talk with each other. The Real Time Protocol (RTP) is such a protocol.

Introduction

The Real-time Protocol (RTP) provides end-to-end delivery services for real-time data stream, such as audio and video. The services provided include payload type identification, sequence numbering, timestamping and delivery monitoring. Typically, RTP runs on top of UDP/IP to make use of its multiplexing and checksum service. But it is not limited to UDP. In fact, RTP is designed to be independent of the underlying transport and network layers. RTP supports both unicast and multicast distributions if they are provided by the underlying network.

It should be pointed out that, although it is called real-time protocol, RTP itself does not provide any mechanism to guarantee real-time delivery. Instead, it relies on the lower layer network services. RTP is a transport layer protocol; it resides between the network layer and the application. But in implementation, it is not applied as a separate network layer. Instead, it is often integrated into the application processing. RTP is not a strictly defined protocol. It is quite flexible, and it is intended to be tailored through modifications and /or additions to the header as needed.

RTP is consisting of two parts:

- The real-time protocol (RTP), which defines how to give packets real-time features.

- The RTP control protocol (RTCP), which defines the mechanism to monitor the quality of service and to convey information about the participants in an on-going session.

Protocol definitions and packet format

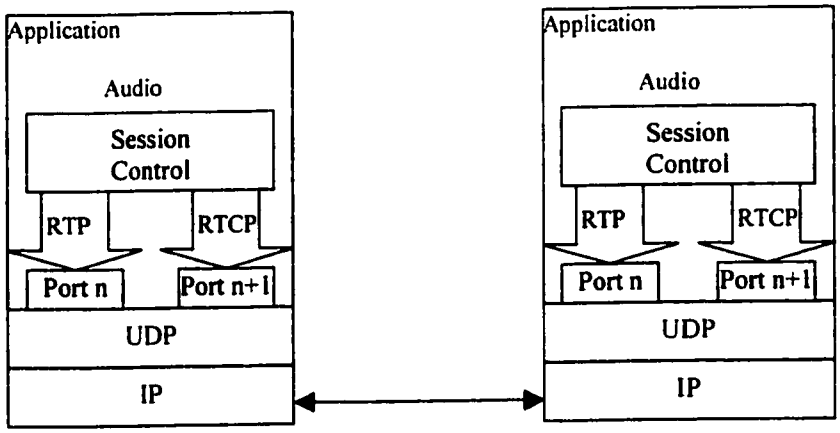


Figure 2.3 RTP used scenario

A typical used scenario is shown in figure 2.3. The association among a set of participants communicating with RTP is called a session. In unicast, there are only two participants. In multicast, the association contains all the participants of a multicast group. For each participant, a pair of ports is allocated to a real time stream. One port is used for RTP packets, which contains data. The other one is used for RTCP associated with the data. In a multimedia session, each real-time stream is carried in a separate RTP session and occupies a pair of transport ports for each participant.

A RTP packet consists of a fixed RTP header, a possibly empty list of contributing sources, and the payload data. Some of the important fields in RTP header are listed as follows:

- Payload type. This field identifies the format of the payload data. An initial set of default mappings is documented in RFC 1890, and may be extended in future RFC.

By using this field, the sender can switch codec dynamically without extra channels to inform the receiver.

- Sequence number. This field is the sequence number of data packets in data stream. It gives a way to detect packet loss and reorder out of order packets.
- Timestamp. This is the most important field in the RTP header. It represents the time of the first octet of the data packet. The timestamp is from a sampling clock that increments monotonically and linearly in time and can be used to synchronize and calculate jitter. The accuracy of the timestamp is determined by the sampling frequency of the source of media.
- SSRC. This field identifies the synchronization source. It is to uniquely identify a synchronization source within a RTP session. It is chosen randomly, and there is a mechanism to detect and resolve collisions.

RTCP packets are sent periodically to all participants. This has four functions. The primary function is to provide feedback on the quality of the data distribution. Such a feedback mechanism is necessary in adaptive bandwidth control. It is also very useful in multicast to let the sender or a third party observer to be able to get the feedback from the receivers to diagnose faults in the distribution. The second function of RTCP is that it carries a persistent identifier for an RTP source. Another RTCP function is that it offers a loosely controlled session control mechanism. RTCP has 5 types of packets. They are sender report (SR) for transmission and reception statistics from senders, receiver report (RR) for reception statistics from receivers, SDES including CNAME, BYE packet to indicate end of participation, and APP packet for application specific functions.

Both the SR and RR include zero or more report blocks. Only the SR packet contains a sender info block. A participant sends SR packets if it sends any data packet during the interval since issuing the last report, otherwise it sends RR packets. Some important fields in sender info block and report block are listed as following:

- NTP timestamp. This timestamp indicates the wall-clock time when this packet was sent. A receiver report with timestamp and elapsed time will be sent in response to this report. The round-trip time can be calculated by combination of these timestamps.
- RTP timestamp. This timestamp represent the same time as the previous timestamp, but in the unit of RTP timestamp (sampling clock). This timestamp in combination with RTP timestamp can be used to do intra-media and inter-media synchronization.
- Cumulative number of packets lost. The total number of lost packets form that given SSRC. It is a factor to represent the quality of delivery.
- Inter-arrival jitter. It is the mean deviation of the inter-arrival time of the packets from that given SSRC. It is a smoothed value, calculated from formula $J=J+ (|D(i-1,i)|-J)/16$, where $D(i-1,i)$ is the difference of transmission time of two consecutive packets. It is another factor to describe the quality of delivery.

The RTCP only offers a mechanism to monitor quality of transmission. It does not define how to analyze the statistics and how to react on the information received. It is up to the application to decide how to use such a mechanism. The implementation of RTP/RTCP is quite flexible. An application can fully realize RTP/RTCP, or only support RTP part. Actually, a lot of applications only use RTP part to make use of its timestamp functionality without any quality monitor function support.

2.4 Voice over IP Applications and Application Protocols

2.4.1 Packet Telephony

Packet telephony is one of the most promising applications of voice over IP. Packet telephony means the ability to make telephone calls (i.e., to do everything we can do today with the PSTN) and to send facsimiles over IP-based data networks with a suitable quality of service (QoS) and a much superior cost/benefit. In order to offer such service, we have to consider not only the QoS issues as mentioned before, but also an infrastructure to conduct telephony services.

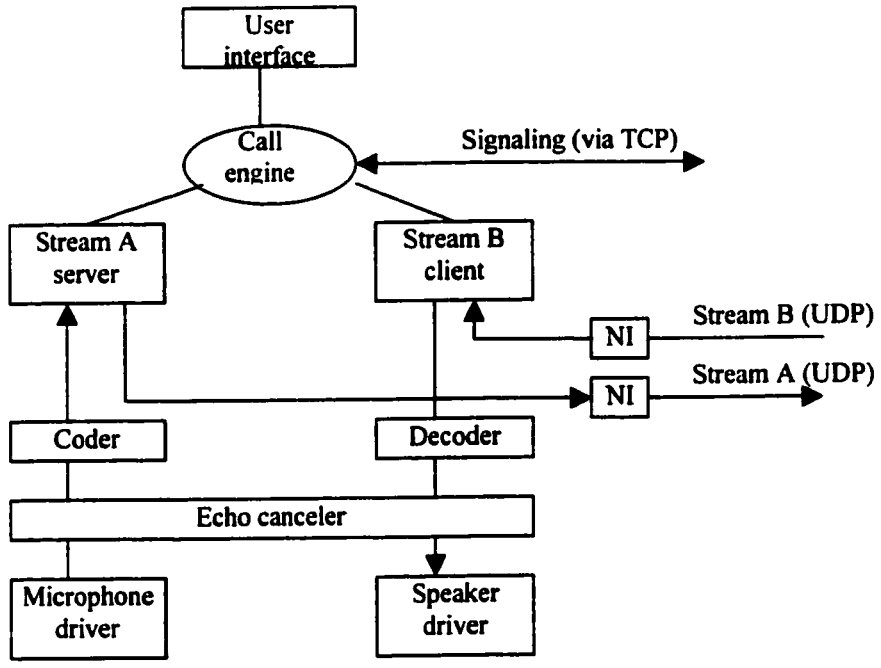


Figure 2.4 End point packetphone architecture

Figure 2.4 shows the structure of end-point of a packet telephony prototype developed by Bell Labs. It comprises a call engine that interfaces to the user for commands and handles all the necessary signaling for call setup/teardown and other functions. The in and out streams are handled by independent stream handlers that take care of network jitter and local scheduling delays and provide continuous streaming of audio. The coder and decoder are separate software components that can be plugged into the audio streams to be able to switch between different codecs. It also contains a software component for handling local acoustic echoes. Because in a packet telephone system the total round trip delay could be much larger than that in PSTN, echo canceling is critical (Echo with large

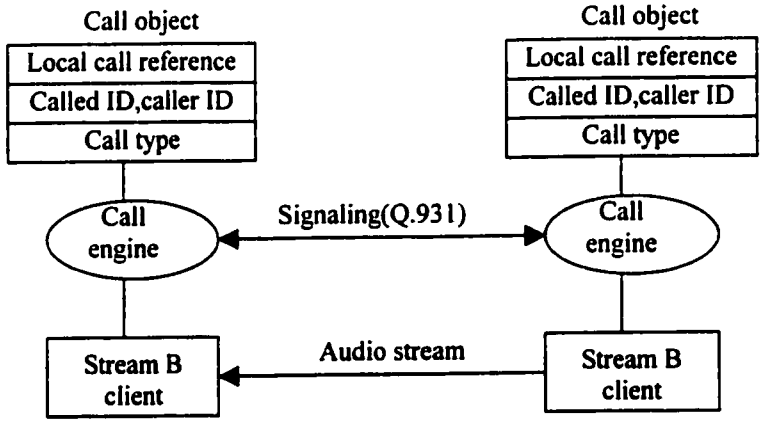


Figure 2.5 Distributed call model

delay is much more annoying than that with small delay). The signaling between two engines needs a reliable signaling channel. The reliable channel is provided by TCP.

In such a packet telephony system, since there is no switch in the middle to hold the state of the call, a distributed call model is applied. This model only supports one end of the call and supports its local state in its call engine and a call object is set up at both ends to support a call. The call object specifies the ID of both parties, necessary information of the connection, and current state of the call.

Packet telephony can be applied over the entire Internet or within a private network. Since in a private network, more control can easily be added to guarantee the QoS, it is more practical to apply packet telephony there. A voice-over-IP gateway is needed in such a scenario. The different scenarios are shown in figure 2.6. The PSTN gateway acts as a bridge between local IP network and the PSTN. The PC-based voice terminal can

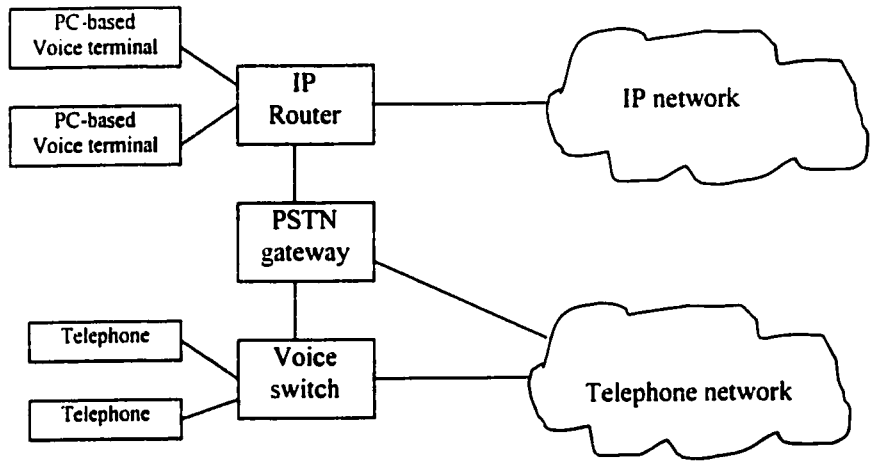


Figure 2.6 Packet telephony infrastructure

make a local phone call via the local IP or access the PSTN through the PSTN gateway.

2.4.2 Advanced Applications

Even though basic telephony and facsimile are the initial applications for voice over IP (VoIP), the long term benefits are expected to be derived from multimedia and multi-service applications, such as multimedia conferencing and Internet commerce. In these applications, audio is an integral part of the systems that may also include video, whiteboard, etc. Currently, there are several multimedia conferencing prototypes developed on Mbone. Mbone is a virtual network on Internet, which supports multicast and group delivery. Examples are Vic and Vat developed by LBL, VeVot by Schulzrinne [5], etc. In these applications, instead of using a complete signaling protocol as

mentioned before, they use some more lightweight control protocols, like SD in Vic and Vat. SD is a session control protocol that can advertise the session and maintain a multicast session.

2.4.3 H.323 Protocol

H.323 is an umbrella recommendation from the International Telecommunications Union (ITU) that provides standards for audio, video, and data communications across non QoS IP-based networks. It is part of a large series of communications standards that enable videoconferencing across a range of networks, known as H.32x. The key benefits of H.323 are:

- H.323 establishes standards for audio and video codec.
- It establishes a mechanism for receiving clients to communicate capabilities to the sender.

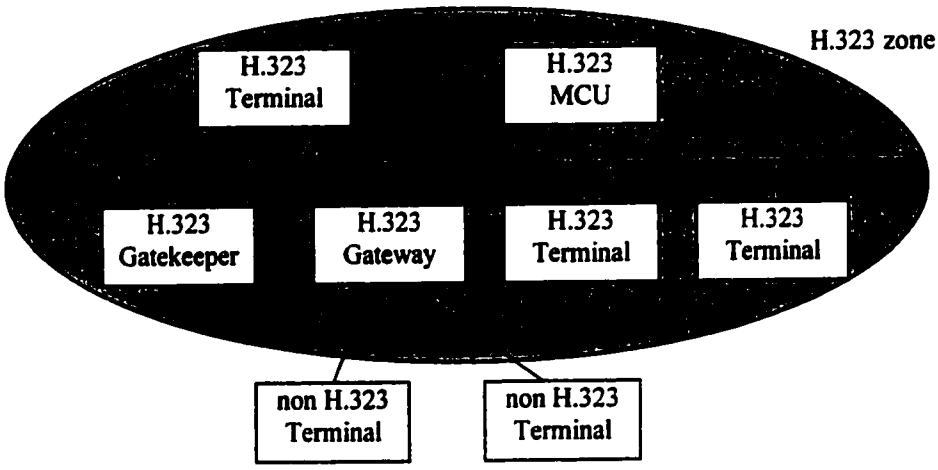


Figure 2.7 H.323 system architecture

- It is designed to run on top of an existing network.
- It is not tied to any hardware or operating system.

- It provides a bandwidth management mechanism.
- Multicast support

H.323 defines four major components for a communication system: Terminals, Gateways, gatekeepers, and Multipoint Control Units. The architecture is shown in figure 2.7 .

- Terminals are end points that provide real-time two-way communications. All terminals must support audio; video and data are optional. Terminals must also support H.245, a protocol used to negotiate channel usage and capabilities. They must also support Q.931 for call signaling and call setup, Registration/Admission/Status (RAS) for communicating with Gatekeeper, and RTP/RTCP for sequencing audio and video packets. T.120 for data conferencing is an option.
- Gateway is an optional element. Its main function is translation between H.323 conferencing endpoints and other terminal types. In addition, it also translates between audio and video codecs and performs call setup and clearing.
- Gatekeeper is the most important component in a H.323 system. It acts as a virtual switch for a H.323 zone. It has two main call control functions. The first is address translation from LAN aliases for terminals and gateways to IP addresses. Other required Gatekeeper functions are admissions control, bandwidth control and zone management. Optional Gatekeeper functions include call control signaling, call authorization, bandwidth management, call management. These control functions are realized by H.245, Q.931, RAS, and RTP/RTCP protocols.
- The Multipoint Control Units (MCU) supports multipoint conferences. An MCU consists of a Multipoint Controller (MC), and zero or more Multipoint Processors

(MP). The MC handles H.245 negotiations between terminals to determine common capabilities for audio and video processing. MP mixes, switches, and processes each audio, video, and/or data streams.

The implementation of H.323 is quite flexible. It can be applied to voice-only handsets and full multimedia conferencing stations. Because of all these benefits, it begins to take root in the market.

Chapter 3 Design Issues of Packet Voice over non-QoS networks

3.1 Structure of a Packet Voice System

A packet voice system that we are going to study can be modeled as in figure 3.1. The

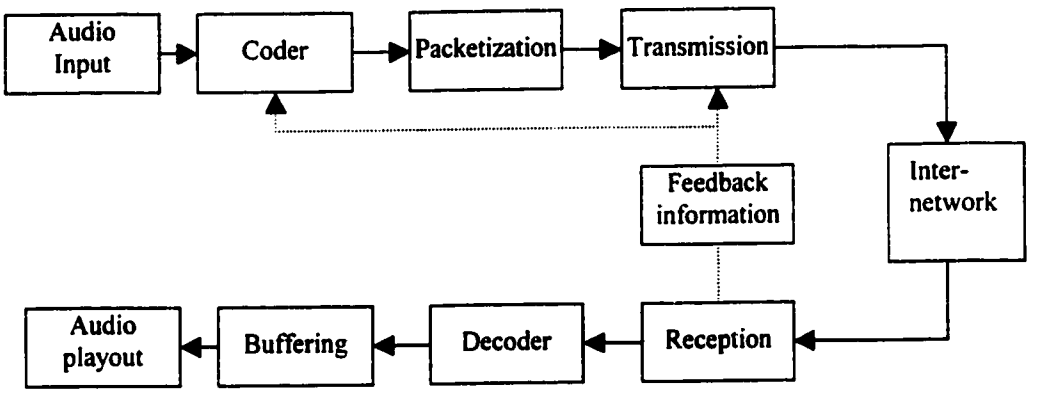


Figure 3.1 Packet voice system

audio input receives analog audio from device, transfers it to digital signal and outputs to codec. The codec component is composed of a collection of codecs that can be switched dynamically. The data then is wrapped in transport layer packets, and a packet header is added for each packet. The header contains timestamp, sequence number, and other real-time related or application related information. We use RTP in our target system. The function of transmission is to receive the feedback from the receiver, apply some transmission control policy to the stream, and push the packets into UDP/IP layer.

In the receiver end, the reception component receives packets from UDP/IP layer and control information from TCP/IP layer, interprets RTP header, reorders the out of sequence packets, sends receiving quality to the sender, and extracts the audio data to

pass to decoder. The decoded data then is put into a playout buffer to compensate the variance of network delay. The data is then passed to the playout device continuously and in the same rate with that of audio input at the sender end.

The network environment between the sender and the receiver is best-effort uncontrolled TCP/IP. There could exist out of order packets, packet loss, and variable packet delay. All these factors are time-varying and unpredictable. A packet voice system over such networks must be designed to compensate these problems to offer an intelligible continuous audio stream, and at the same time control the end-to-end delay under an acceptable level. We will introduce some strategies of bandwidth control and packet loss recovery in the following two sections. But our focus is on delay jitter management, because we think delay jitter is the most important factor that affects the quality of audio, and the jitter control strategy is the most effective compensation method compared with bandwidth control and packet loss recovery.

3.2 Bandwidth Control

3.2.1 Introduction

Network congestion is the main reason that causes packet loss and packet delay. The traffic that causes the congestion may be from audio, or from other data transmission, or from both of them. The packet loss or packet jitter caused by short term congestion can be compensated by some adaptation methods. But these methods don't help when the congestion is long and persistent. In long term congestion, the throughput of frames between a transmitter and a receiver may not be sufficient to sustain normal display rates.

In this case, a bandwidth control strategy comes in to play a role. The idea of bandwidth control is to minimize packet loss by controlling the send rate at the sender end according to the feedback information received from the receiver end. It should be pointed out that such a control strategy is not always effective. The reason is that not all the sources use the bandwidth control strategy. So, it is possible that even if an audio source decreases the sending rate, it still finds the loss rate or other receiving qualities to be the same. This is the limitation of the strategy.

In order to support such a bandwidth control strategy, we must be able to 1) adjust the rate 2) have a feedback mechanism. We have some difficulty with the first point in that there does not exist a codec, whose output rate can be controlled over a wide range of bit rates with a relatively fine granularity. This problem can be partly solved by using a series of codecs with different bit rates. Although the granularity is coarse, we can at least adjust the rate close to that desired. For point 2), we already have RTP/RTCP, which can give QoS feedback. As introduced in section 2.3.4, in a RTCP receiver report, there are fields of 'cumulative number of packet loss' and 'inter-arrival jitter' which can describe the quality of delivery. This mechanism scales well from unicast to a large multicast group. In multicast situation, the source must convert the receiver reports from different receivers into a global QoS measure, which represents the overall quality.

3.2.2 Control Algorithm

Here we describe a control algorithm, which is used in an INRIA project [39]. This algorithm is quite simple, while easy to implement. This algorithm gradually increases the send rate at the source if the loss rate is below a threshold. It decreases the send rate if

the loss rate is above another threshold. It selects a set of codecs, which give different output bit rate. These codecs are PCM, ADM6, ADM5, ADM4, and GSM. The experiment shows that this scheme works perfectly in a controlled local network environment. In this environment, audio is the main source of traffic. The quality only decreases slightly when the congestion increases. But when moving it to a wide area uncontrolled network, the algorithm shows its limitation. The reason is obvious. In the former situation, when most terminals use the strategy, they still share the bandwidth fairly when congestion increases. But in the later situation, the bandwidth is not under control at all. The scheme does not help to increase the bandwidth, but only to adapt to that available bandwidth. When combining this scheme with packet loss recovery methods, the quality of delivery can be increased further. We will introduce the methods in the next section.

3.3 Packet Loss Recovery

3.3.1 Packet Loss Behavior in the Internet

Packet loss is one of the main reasons that affect the quality of audio. In order to design recovery algorithms, it is critical to understand network packet loss behavior. Because it is so difficult to model the loss pattern, the best way to study this problem is by experimenting. In [37], experiments are carried out between two Internet nodes with audio packets upon RTP/UDP/IP. Each audio packet contains 40 ms of speech and packets are sent periodically every 40 ms. The packet loss is measured in two aspects. The first characteristic is the average packet loss, which is the expected value. But this

parameter is not enough to characterize the burst feature of packet loss, which is the most

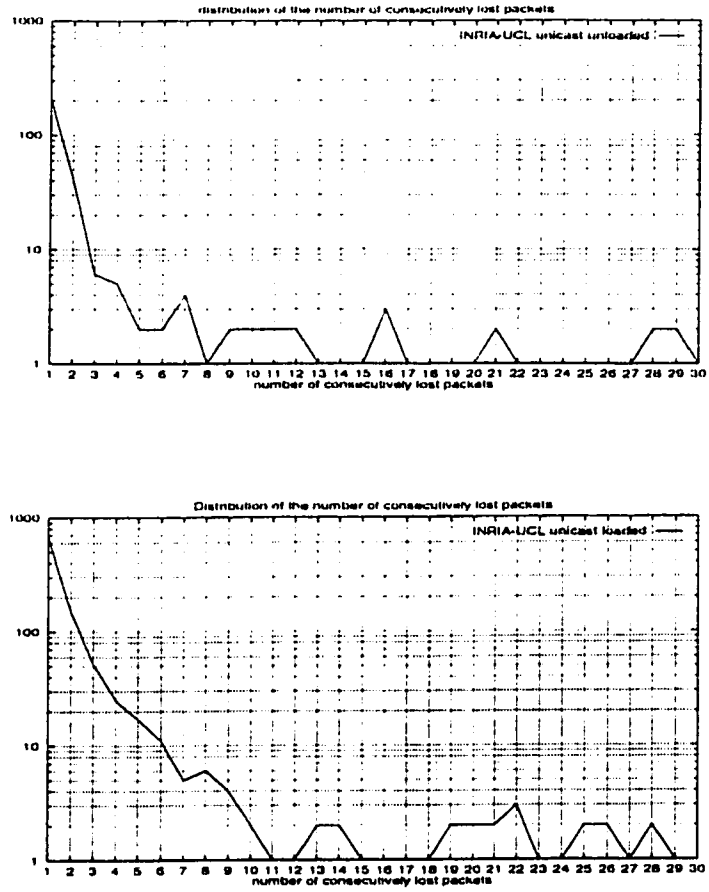


Figure 3.2 Distribution of the number of consecutive lost packets at 8:00 am (top) and 4:00 (bottom)

important feature of packet loss. So the second characteristic is the expected number of consecutively lost packets. The measure was performed at 8:00 am, when traffic is light, and 4:00 pm, which is the peak hour. The results show that 1) most of the packet losses are isolated except some burst periods, 2) the distribution of consecutive packet loss is similar in light traffic and in heavy traffic, but the consecutive packet loss rate in heavy traffic is much higher than that in light traffic, and the length of loss is also much higher. The result is shown in figure 3.2. The curves are the number of occurrences versus the

number of consecutive losses. Considering the curves are drawn on a log scale, most of the packet losses are isolated.

3.3.2 Packet Loss Recovery Algorithms

The aim of recovery is to construct a suitable dummy packet at the receiver, so that the loss is as imperceptible as possible. The algorithms can be split into two categories; receiver only, and combined source and channel techniques.

The receiver-only techniques are those that recover the missing packets only at the receiver end by filling the gap with something that tries to make the loss imperceptible. The simplest one is silence substitution. Research shows that it gives adequate performance when the packet is small (<16 ms) and the loss rate is low (<1%) [14].

Another receiver-only technique is white noise substitution. White noise is shown to give performance improvement over silence when contextual information in speech is removed, and intelligibility improvement when the contextual information is present. The reason is that the ability of the human brain to subconsciously repair the missing segment of speech with the correct sound is stronger in the noise situation than in the silence situation. Because white noise substitution is as easy to implement as silence substitution, while it gives better performance, it is considered as a better choice. There are some other receiver-only techniques that are based on the assumption that the missed speech segment doesn't change very much from the previous segment. These techniques use recent received packets to prospect the missed packets. A simple example of these techniques is repetition of the previous packet. But like the silence substitution, all the receiver-only

recovery techniques have the limitation that they only work well when packet size is small and the loss rate is low.

Combined source and channel techniques have much better performance than receiver-only techniques. Automatic Repeat Request (ARQ) is one of these techniques. A feed back mechanism is build between sender and receiver to let the sender retransmit the packets that were not received at the destination. This mechanism requires an extremely large buffer time at the receiver end, which is unacceptable in many applications. Forward Error Correction (FEC) is another one of this kind of techniques. It is based on adding redundant information into packets, and it is proved to be a reconstruction algorithm that gives high performance and robustness to high loss rate.

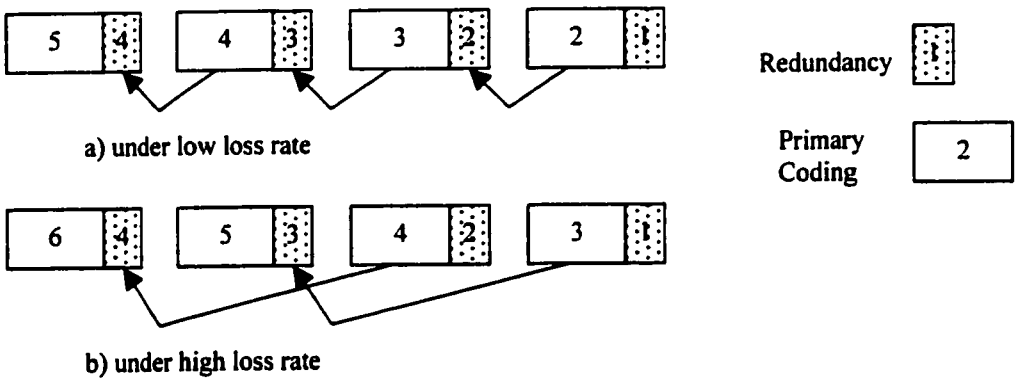


Figure 3.3 Forward Error Coding algorithm

The algorithm uses two audio codecs for the same audio stream, one for primary coding, and the other for redundancy coding. The codec for primary coding has a relatively high bit rate (like PCM or ADPCM), and the codec for redundancy coding has low bit rate, (like LPC). The redundancy information is piggy-backed into the following packet that contains the primary speech. So, isolated packet loss can be recovered by using the redundant information in the following packet. The algorithm can be extended that the redundant information of packet n is piggy-backed into packet n + d, where d can be 1, 2,

... as shown in Figure 3.3 b), where d=2. In this way, consecutive packet loss can be recovered. Since, as we know, consecutive packet loss happens more often in high traffic load, a feedback mechanism can be used to let the sender switch between different redundancy schemes to adapt to different traffic load. But, obviously, this brings the problem of long buffering time in the receiver end.

3.4 Delay Jitter Management

3.4.1 Introduction of Jitter Management

In network transmission, the display latency of a packet is defined as the total time from acquisition at the sender to display at the receiver. This period of time can be divided into two parts: end-to-end delay (the total time from acquisition to decompression) and display queuing delay (the time spent in buffer). As we introduced above, because of the variance of network traffic over time, different packets experience different transmission time. The variance in end-to-end delay is called delay jitter.

The delay jitter is the main reason that affects the quality of audio. Experiments show that, compared to video, audio is extremely sensitive to display discontinuity. For example, a gap caused by a late packet arrival of video can be filled with its previous packet without any adverse effect. But this does not occur to audio. Experiments show that a discontinuity of only 16.5 ms has a significant effect on human perception. In order to play the packets continuously and at the same rate as they were recorded, the jitter must be compensated. It is obvious that the jitter can be compensated by varying the display queuing delay (figure 3.4).

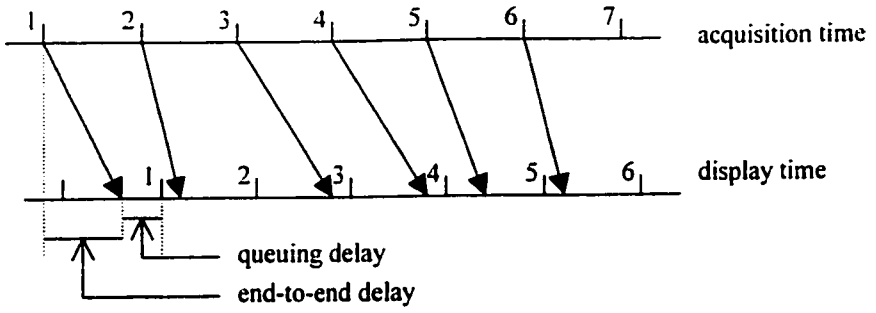


Figure 3.4 Delay jitter compensation

Theoretically, the jitter can be fully compensated by considering the worst-case end-to-end delay. But it is not so simple. The problem is that choosing the largest possible jitter means extremely long display latency, which is not acceptable in many real-time applications. And because of the unpredictability of the network, it is difficult to find such a largest jitter. So, normally, we would not choose such a largest display latency, which can compensate all the jitter. Instead, we choose a display latency large enough to compensate most jitter. In this case, as a result, there is still a chance that when a frame incurs a particularly long end-to-end delay, the frame may not be ready to be displayed at the time when the display of the preceding frame is complete. There will be a gap in the playout. It is proved that in most conferencing applications, as long as gaps occur infrequently, playout with low latency and some gaps will be better than playout with high latency and no gaps. In general, the lower the display latency, the higher the probability of encountering a gap. So, a tradeoff must be made between display latency and gap frequency.

Theoretically, in order to choose appropriate display latency, we must have a good understanding of network end-to-end delay behavior. End-to-end delay in the Internet has been widely studied with analytic, simulation and experimental approaches. Some of these researches indicate that the delay distribution can be modeled by a constant plus

gamma distribution, where the parameters of the gamma distribution depend on the path and the time of the day. The studies also show the behavior of short time scale (which is more important to jitter smoothing for packet voice) is different from that of long time scale [40]. The most important behavior of short time scale is that the typical end-to-end delay pattern is long periods of low delay separated by bursts of high delay. In another research [33] on a LAN environment, the result shows that the end-to-end behavior has characteristics that the standard deviation of delay is normally larger than the average delay, and when traffic increases the standard deviation increases much faster than the average delay. Generally, due to its complexity and diversity, the end-to-end behavior is difficult to characterize, so as to serve as a guide in designing a delay jitter smoothing strategy. In designing jitter management algorithms, we can have some general knowledge of end-to-end delay as: 1) The end-to-end delay can be modeled by a constant plus gamma distribution, which means the maximum delay is much larger than the average delay and is much less frequent. 2) The delay has bursty characteristic, which means the long delay always comes in bursts. 3) With traffic increasing, the deviation of delay increases faster than the average delay. It was also observed that the distribution of delay becomes flat when the traffic load is high. In a practical application, in order to get knowledge of current network conditions, we often use a recent observation as a pessimistic estimation of the near future.

If we accept packet gaps, there arises another problem: how to deal with the late packets. The problem of jitter management actually becomes how to choose a display latency to make the best tradeoff between latency and gap frequency, and how to deal with the late packets.

3.4.2 Buffer Management Algorithms

Most jitter management algorithms choose display latency by observing the recent delay history. Here we describe such an algorithm[40]. There are two assumptions for this algorithm. First, we assume that the clock in the sender and the clock in the receiver are precisely synchronized. Under this assumption the delay can be calculated by the receiver from the timestamp attached with the packets it received. Later, we will discuss the problem of synchronization. The second assumption is that we are able to change the

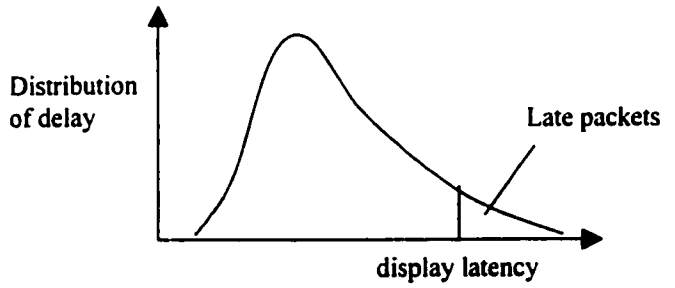


Figure 3.5 choosing display latency

latency during the voice communication. If the audio is human conversation, it can be naturally divided into active periods and silent periods. We called the active periods as talk-spurts. The beginning of a talk-spurt provides a good chance to change display latency. As long as the change of display latency is within a limit, it doesn't affect the quality of the output audio.

In a jitter management algorithm, the main problem is the adaptive setting of the display latency. Unless the latency is very large, there is nonzero probability that a gap will occur. We would like to choose latency that can control the gap probability within a threshold, and at the same time keep it as small as possible. (Experiments show that when 6 percent of packets caused gaps, the quality is considered acceptable [40].) We keep the

delay of recent m received packets and use it as a pessimistic estimate of the current delay distribution. We then choose the display latency from these delay values. We select the k th largest from the m values as the new latency. The m should be large enough to make the estimation close to the current network state. At the same time, we would not choose a too large m in order to, first, eliminate isolated cases of extremely large delay, second, save computing resource spent in recording and analyzing the delay history. The value of k determines the probability of gap frequency. The larger the k, the larger the gap frequency. We select the k based on a given gap frequency requirement.

Once the display latency is selected, the remaining problem is how to deal with late packets. The first method is to wait for late packets forever, and as a result the display

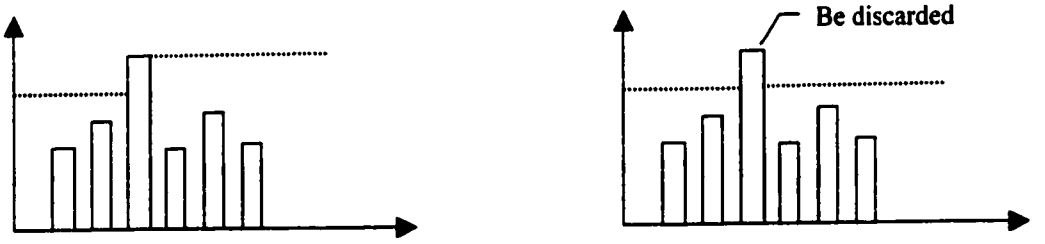


Figure 3.6 E-policy and I-policy

latency is expanded. This method is called E method in [40]. The other method is to discard any late packets to keep the display latency unchanged. This is called I (ignore) method. There is a third method, which is between the above two methods. This method sets a threshold of waiting time. If packets arrive within this threshold, they will be played, and the display latency is expanded. If packets arrive later than the threshold they are discarded. So, with this method the display latency could be expanded but within a limit. We call this method as threshold method.

It is obvious that the I method will give us the minimal delay. But it loses packets. The E method tries to play all the received packets, and as a result it produces much larger

display latency. Suppose we only adjust the display latency at the beginning of talk spurts. With the *I* method, once we select the display latency when the first packet of a talk spurt is received, the latency will be kept constant until the end of the spurt. But with the *E* method, the latency is expanded whenever a packet with latency larger than the current latency is received. We can anticipate that after some time the display latency will be close to the largest end-to-end delay. The latency can only be reduced at the beginning of the next talk spurt. While the threshold method makes compromise between the above two methods, its performance largely depends on the selection of the threshold value. Actually, by setting the threshold to 0 and infinity, we have the *I* method and the *E* method respectively. So the *I* method and the *E* method are special cases of the threshold method.

A theoretical analysis was given in [40]. In the analysis, it was assumed that the display latency was chosen at the beginning, and was not adaptively adjusted. The network delay has an exponential distribution. The tradeoff of delay latency and gap frequency was evaluated for the *E* policy and the *I* policy. Given the quality of 6 percent gap probability, it is concluded that choosing m as 40 and k approximately 7 percent of m for the *E* policy and 5 percent for the *I* policy can give the best performance.

3.4.3 Some Other Jitter Management Algorithms

One assumption of the above jitter management algorithm is that the latency can be adjusted at the beginning of each talk-spurt. But there are some situations, in which the audio stream can not be divided into active periods and idle periods, such as music. In this case we should treat the audio stream as a continuous flow and we must use some

other strategy to manage the display buffer. The other disadvantage of the above algorithm is that it requires a synchronized clock. Below, we introduce some algorithms that are suitable for this situation. The first is the queue monitoring algorithm [41].

Queue monitoring

In this algorithm, instead of measuring end-to-end delay, we measure the length of the display queue, which is another indicator of network jitter. Every frame time, a packet should be moved from the queue to the playout device. At the same time, if the end-to-end delay is constant, there will also be a packet added to the queue. In this case the length of the display queue is constant. But when there exists jitter, the situation is different. If end-to-end delay increases sufficiently, it is possible that no packet arrives during a frame time. In this case the queue length decreases. On the other hand, if the end-to-end delay decreases, more than one packets may arrive and are put into the display queue during a frame time. Then the queue length increases. So by observing the length of display queue, we can have the knowledge of recent network congestion. If we assume the level of delay jitter in the near future will be the same as the level in the recent past, then we can use the knowledge we gained by recent observations in order to manage the queue.

The basic idea of this method is setting a threshold of the queue length and discarding packets when the threshold is exceeded. We define a threshold value for each possible display queue length. The threshold value for each possible queue length n specifies a duration after which, if the display queue still maintains more than n packets, we discard the oldest packet. To implement this strategy, we set an array of pairs of a threshold and a

counter associated with the threshold. When it is the time to play a frame, the following operation is applied. First we check the length of the queue. If the length is *m*, we increase the counter 2 to m-1 by 1 and reset all the other counters. If any counter exceeds the threshold associated with the counter, the oldest frame is discarded and all the counters are reset. To prevent an empty display queue, the policy does not apply to a display queue with length of 2. The reason is, when only two packets are left in the queue and one is discarded and the other is played, the display queue will be empty. If the next frame has a slightly larger end-to-end delay, then no packets arrive during the next frame time. In this case a gap is generated.

This algorithm is designed to deal with bursts of network traffic. Since network traffic

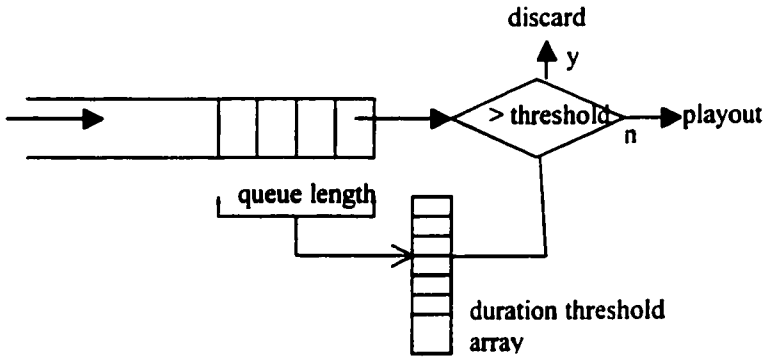


Figure 3.7 queue monitoring strategy

has the property that large delay variations occur less frequently, and small variations occur more frequently, we set the threshold value in such a way that the duration of long queue length is short, and the duration of short queue length is longer. The strategy is claimed to handle burst traffic properly. When network traffic increases and end-to-end delay increases, the display latency is expanded. When the burst is over, the display queue will increase. The algorithm detects the increase of queue length and starts to decrease the display latency by discarding packets. After some time the display latency

will return to the minimum latency. This algorithm is also a flexible algorithm. Its performance is largely affected by the setting of the thresholds of duration. By changing the thresholds, we can have low latency with frequent gaps to high latency with few gaps. Just as the threshold policy introduced in section 3.3.3, we can make use of its flexibility to meet different requirements in different applications.

Holding time

One of the problems of the queue monitoring algorithm is that queue length is a coarse indicator of delay jitter, especially when the packet size is large. An alternative method is to measure the holding time (buffering time) of each packet. Like queue monitoring algorithm, it also doesn't require a synchronized clock. A holding-time adaptation algorithm was introduced in [43]. In this algorithm, the holding times of received packets are recorded, and a short-term average holding time is maintained. This algorithm first defines a target mean holding time H_t, which is based on an estimate of current network condition. For each packet to be played, if it arrives after its desired playout time, it is discarded. Periodically the actual average holding time Ha is calculated and compared with the target holding time. If the absolute value of Ht - Ha is greater than a target error, then the holding time of the current packet is increased or decreased by expanding or compressing the playout time of the current packet. (It should be mentioned that small variation of playing speed doesn't affect the audio quality very much.) In this way we make the actual holding time approach the target holding time. An improvement has been made on the above algorithm to make the target holding time adaptive to network traffic. The packet losses are monitored and evaluated periodically. If the loss exceeds a preset

threshold, the target holding time is expanded and the expansion step size is chosen based on the severity of packet loss.

3.4.5 Time Synchronization & NTP

Clock synchronization is an important issue in delay jitter management. The first jitter management introduced in 3.4.2 is based on a synchronized clock between sender and receiver. In this algorithm, we use a local clock - timestamp attached with each packet to get the delay. In the 'queue monitoring' algorithm and 'holding time' algorithm, synchronization of clock is not necessary. But we will find that it still requires that the sender clock and the receiver clock run at almost the same frequency. Actually from the following analysis we will find that even for the first algorithm, there is no need for the clocks to be synchronized. The really important thing is that the clocks run at the same speed.

Let s(i) be the timestamp of packet i, and d(i) be the arrival time of packet i. So s(i) is measured in sender clock, and d(i) is measured in receiver clock. If the two clocks are synchronized, we use the following formula to calculate the delay. Let u(i) be the delay of packet i.

$$u(i) = d(i) - s(i) \tag{3.1}$$

$$u(j) = d(j) - s(j) \tag{3.2}$$

$$u(j) - u(i) = d(j) - d(i) - (s(j) - s(i)) \tag{3.3}$$

If there is a difference of e between the two clocks but they have the same frequency, then the delay can be expressed as

$$u(i) = d(i) - (s(i) + e) \tag{3.4}$$

$$u(j) = d(j) - (s(j) + e) \tag{3.5}$$

But,

$$u(j) - u(i) = d(j) - d(i) - (s(j) - s(i)) \tag{3.6}$$

Therefore the difference of clocks doesn't affect the variance of delay. We can estimate a network delay for one packet (i.e. the first packet) and correlate all the other packets to that packet. The estimation could be based on general knowledge or a time reference. As long as the two clocks run at the same speed, the clock synchronization is not required for the analysis of delay jitter.

But if there is a difference of frequency between the clocks, it will bring problems to buffer management. Because the difference is cumulative, even if it is only a slight difference, it will cause trouble. Let's see the first algorithm. We assume the frequencies of two clocks have a slight difference, and for each frame time the receiver clock has a slight offset of f over the sender clock. If at the time of packet i the difference of the two clocks is e, then:

$$u(i) = d(i) - (s(i) + e) \tag{3.7}$$

After n packets, the delay can be expressed as

$$u(i+n) = d(i+n) - (s(i+n) + e + n \times f) \tag{3.8}$$

The delay variance should be

$$u(i+n) - u(i) = d(i+n) - d(i) - (s(i+n) - s(i)) - n \times f \tag{3.9}$$

Compared with formula 3.3, the difference of frequency caused a delay variance of $n*f$. With the increasing of n, a small f will make a large difference. If the receiver clock is a little bit slower than the sender clock, the display queue will tend to be in starvation state even when the network traffic is constant. Otherwise, the display queue will have a

tendency to be in overflow state. Without a synchronization mechanism the receiver can't tell whether the delay variance is caused by network traffic or by the speed difference of the clocks. The problem also exists for the queue monitoring algorithm and the holding time algorithm.

NTP (Network Timing Protocol)

The solution is to build a clock synchronization mechanism in the network. The purpose is to correlate the receiver clock with the sender clock. NTP (network timing protocol) provides such a mechanism. NTP and other synchronization protocols, like DTSS, are designed to provide time service to network computers without time receiver equipment, such as GPS. Some workstations that have direct access to time equipment act as primary time servers, and clients and second servers use synchronization protocols to correct their clocks to the server clocks. The synchronization mechanism can provide accuracy within a millisecond on LANs and up to a few tens of milliseconds on WANs.

The primary model of NTP is a client-server model. On request, the server sends a message including its current clock value or timestamp, and the client records its own timestamp upon arrival of the message. The protocol measures the one-way delay by measuring the roundtrip time. It assumes the propagation times are statistically equal in each direction. The second model supported by NTP is symmetric mode, which allows either of two peer servers to synchronize to the other, in order to provide mutual backup.

The accuracy of one way time estimation is affected by several factors. The key factors are network jitter and different paths between two peers in a WAN. NTP has defined some strategies to deal with these problems. NTP has redundancy time servers across the

networks. For each peer the packets with timestamp are transmitted periodically. The most accurate clock offset is measured at the lowest delay. NTP not only measures the clock offset, but also measures the frequency offset. The current NTP version is version 4.

3.5 Other Issues of Packet Voice System Engineering

3.5.1 Packet Size

The other aspect that we have to take into consideration in a packet voice system is packet length. When investigating the total display latency we find that it is composed of a packetizing delay and end-to-end delay (We don't consider buffering delay in this analysis). The packetizing delay actually is the duration of a voice packet. The end-to-end delay is the delay we studied in the previous sections. Given an audio stream, its packetizing delay increases linearly with the increasing of packet size. Looking at the

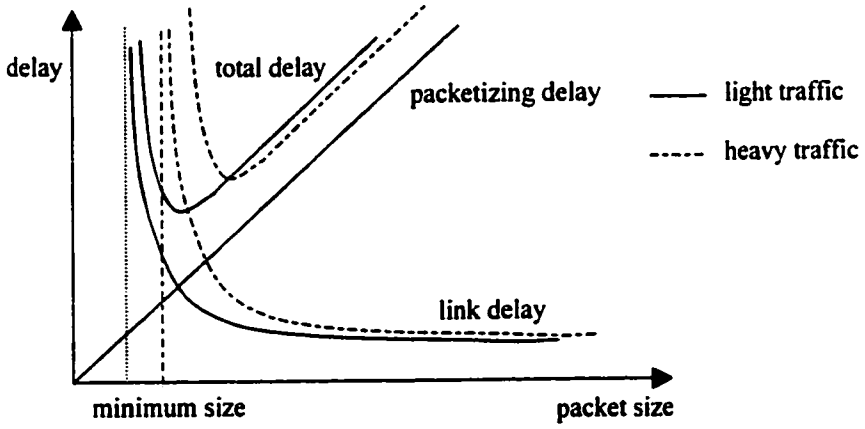


Figure 3.8 Total delay vs. packet length

relation of end-to-end delay and packet size, we understand that with the increasing of

packet size, there is less average overhead on a unit audio section. The overhead includes packet headers and the network resources needed to transfer packets (considering the shorter the packet, the more packets generated). The curve of end-to-end delay versus packet size is not linear. Because there exists a bottleneck for a given link, there is a minimum packet length, when the packet size is approaching this point, the packets would block the link and the end-to-end delay would increase sharply. The situation is shown in figure 3.8. From the curve of the total delay, there is an optimum packet size, which can give the smallest total delay. It can be anticipated that with the increase of network traffic, the curve of link delay will move to the right, therefore the curve of total delay will move to right and up. This means the optimum packet size increases with network traffic.

But the end-to-end delay is not the only consideration in designing a packet voice system. Actually, in order to have better efficiency, we often choose a larger packet size, as long as the requirement of display latency can be fulfilled. The above analysis just gives us a guide on choosing the packet size.

3.5.2 Desktop Scheduling Problem

Desktop real-time applications require deadline operations, such as removing a block of data from the application to a device. Missing the deadline will result in frequently

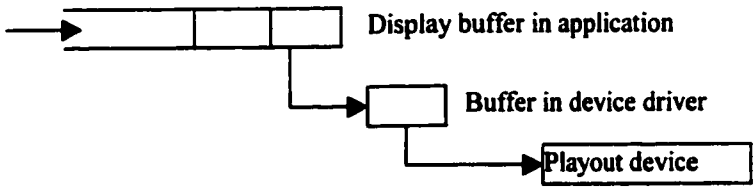


Figure 3.9 Data flow in receiver

disrupted audio. Traditional time-sharing operating systems for general purpose workstations do not provide real-time service for packet voice applications. A solution to provide QoS guarantees is to modify the operating system scheduler to provide a bounded dispatch latency for applications. Although this can significantly improve the quality of packet voice applications, it has not been implemented in popular operating systems. Similar to the application adaptation for network jitter, there is application adaptation for this operating scheduling problem.

In a desktop packet voice application, there is a strict loop operation. In each loop, the receiver moves a block of data from the display buffer to the device driver. Then the block of data is saved in the buffer of the device driver for playing in the next period of time. In a Windows NT system, when the device driver finishes with playing the data in its buffer, it calls a callback function to move the next block of data, or just sets a flag and the application polls the flag to detect the state of playing and feeds the next block. But on a multitask time-sharing operating system, keeping with the loop may not be always possible. The scheduler decides when an application can have the access to the CPU. If, because of other applications running, the audio misses the point when a new block of data should be fed into the device driver, a gap is generated in the output. Measurements in an overloaded workstation show that the delay variance can be as much as several milliseconds.

To compensate the scheduling variation, we can put extra data into the device driver buffer. If the extra data is long enough, it can compensate most scheduling delay. Like the application buffering, the extra buffered data in the device driver buffer will cause extra display latency. There is also a tradeoff between gap frequency and display latency.

The extra data is called cushion. It is desired to choose an appropriate cushion size based on the current workstation state. By observing the recent scheduling variance, we can have an estimate of cushion size. When moving the first block of data, we move an extra block of data into the device driver buffer as cushion. In the following cycles, we just maintain the cushion level. If portion or all of the cushion is consumed because of a scheduling anomaly, we refill the consumed part. The cushion size is updated periodically according to the recent observed scheduling variance, and the updating is performed at the beginning of a talk-spurt.

3.6 Chapter Summary

In this chapter, we have studied application adaptation problems, and provided solutions. The adaptations are in the areas of bandwidth control, packet loss recovery, delay jitter compensation, desktop scheduling compensation, packet length consideration, etc.

Bandwidth control provides perfect performance in a controlled LAN environment, while it has its limitation in an uncontrolled WAN environment. Several packet loss recovery algorithms are reviewed, with focus on the redundancy algorithm. Receiver-only recovery techniques are easy to implement but fail when the packet is long or the loss rate is high. The redundancy algorithm has better performance, and is adaptive to network traffic. The disadvantage of this algorithm is that it introduces extra display latency.

It is considered that jitter is the main problem of packet voice over a non QoS network, and jitter management strategy is critical to application adaptation. In this chapter, the network delay jitter problem is studied, and jitter compensation algorithms and clock synchronization methods are discussed. The key problem in jitter management is to make

balance between gap frequency and display latency. We design a jitter management algorithm by managing the balance and adjusting latency according to current network conditions. The desktop scheduling problem is similar to the network jitter problem, and the solution is also similar.

Chapter 4 Simulation Study of Jitter Management

4.1.Introduction of Simulation

Considering jitter management is the central problem in application adaptation, we developed a simulation platform to implement, test and evaluate the performance of different jitter management algorithms. The application was written in Visual C++ on Windows 95.

In this application, we assume the clocks of sender and receiver are precisely synchronized. We also assume there is no packet loss in the transmission. To concentrate on network delay jitter compensation, we don't simulate desktop scheduling problem and other problems.

The application reads audio data from a wave file. The wave file is in either 8 bits or 16 bits PCM format. A packetizing block is responsible for putting data into a packet, and

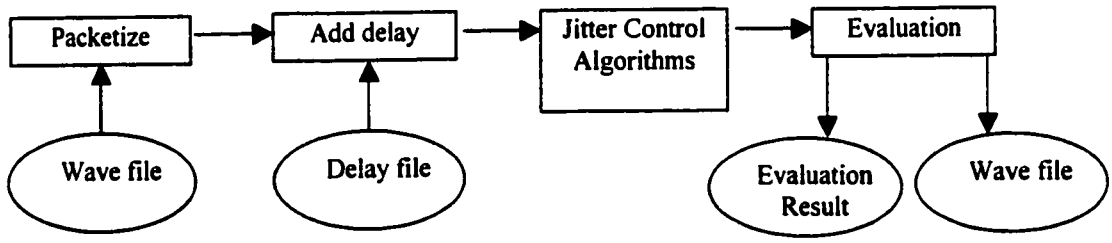


Figure 4.1 Structure of simulation Tool

wrapping the packet into a packet header. We use a RTP header format, and a timestamp is recorded into the header. The timestamp is measured in sampling clock periods. In the add-packet block, a delay is generated for each packet to simulate a real network situation. Two modes are provided to generate delay. In the first mode, the user can configure an exponential distribution, and delay is generated dynamically according to an

exponential distribution. In the second mode, delay is read from a delay file, and we use other tools to generate the delay file. A network end-to-end delay model is provided to generate delay files. Users may also use other delay models to generate delay files, or they can use the delay recorded from a real network. The packets together with delays associated with the packets are passed to the jitter management block. The jitter management algorithms implemented in this application are based on the algorithms introduced in section 3.3.2. They are E-policy, I-policy and Threshold policy. The application can switch between these algorithms. The parameters of these algorithms can be configured at the beginning of the simulation. The output of the jitter control algorithm is passed into the evaluation block, where the audio is recorded into another audio file just as it would be played in an output device. A set of parameters is calculated to evaluate the quality of the jitter management algorithms.

4.2 Interface

The application was developed in Visual C++. A GUI is offered to configure the system and to set the experiment's environment. In the main menu, the user can launch the configuration window, start simulation, stop simulation, or transfer packets to other network nodes (but we don't do any distributed experiment). All the parameters for the configuration can be set in the configuration window. They are listed as follows.

Packet size (in bytes),

Packet interval,

Input wave file name and path,

Output wave file and path,

Parameters for dynamic jitter generator,

Whether input jitter is from file (if yes, the file name),

Type of jitter management algorithm (E, I , Threshold) and parameters for each algorithm.

The application also provides a monitoring window to dynamically display the running condition of the ongoing simulation.

4.3 Packetizing Block

We simulate the audio input by reading from an audio file, and generate packets with an interval of a frame time. We use a wave file. The format of wave file is shown in figure 4.2. It is composed of a series of chunks. The first chunk is the format chunk, which

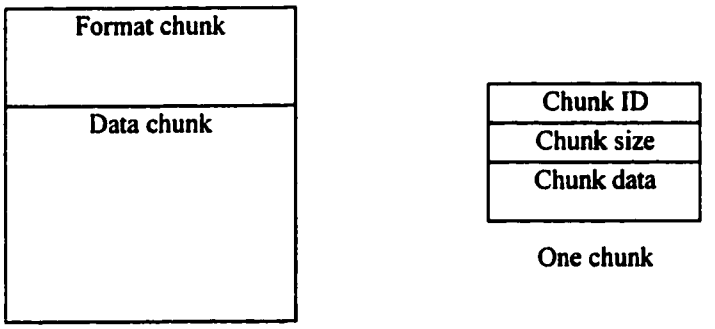


Figure 4.2 Format of wave file

contains information to interpret the audio data, such as sampling rate, channels, sampling bits (16 bits or 8 bits), how many octets per second, etc. The data chunk contains the data stream. Each chunk has its own header, which give the ID and length of the chunk. When reading the file, we first read the format chunk, and then get the information of the chunk, then we can start to read the data chunk. In this application, we also need to pass the format chunk to the other blocks to interpret the data stream.

Periodically a packet is generated and passed into next block. We use the RTP packet format to wrap the audio data. The functions that are related with the RTP packet operation are part of a H.323 API package from Databeam. In the RTP header, we only care for the timestamp field. The meaning of the timestamp is sampling time of the first octet in the packet. In this application, we use a software clock to synchronize all the components of the application. We make the soft clock running in sampling tick. Then, we just fill the timestamp with the current value of the soft clock.

4.4 Jitter Management Block

The jitter management block receives packets and delays associated with the packets from upstream. Three algorithms are implemented. They are based on E-policy, I-policy, and Threshold policy. In the configuration stage, one of these algorithms is selected, and parameters for that specific algorithm are also set. The flow diagram is shown in Figure 4.3 a) and b).

4.4.1 Data Flow

The whole application is synchronized by a software clock. The clock runs in sampling rate. In each tick, the process goes through the buffer management process and performs the algorithm.

In the jitter management block two queues are managed. One queue is designed to simulate the display buffer. The other queue is to save packets that are generated but not arrived. Periodically, the packetizing block generates packets and passes them to the jitter

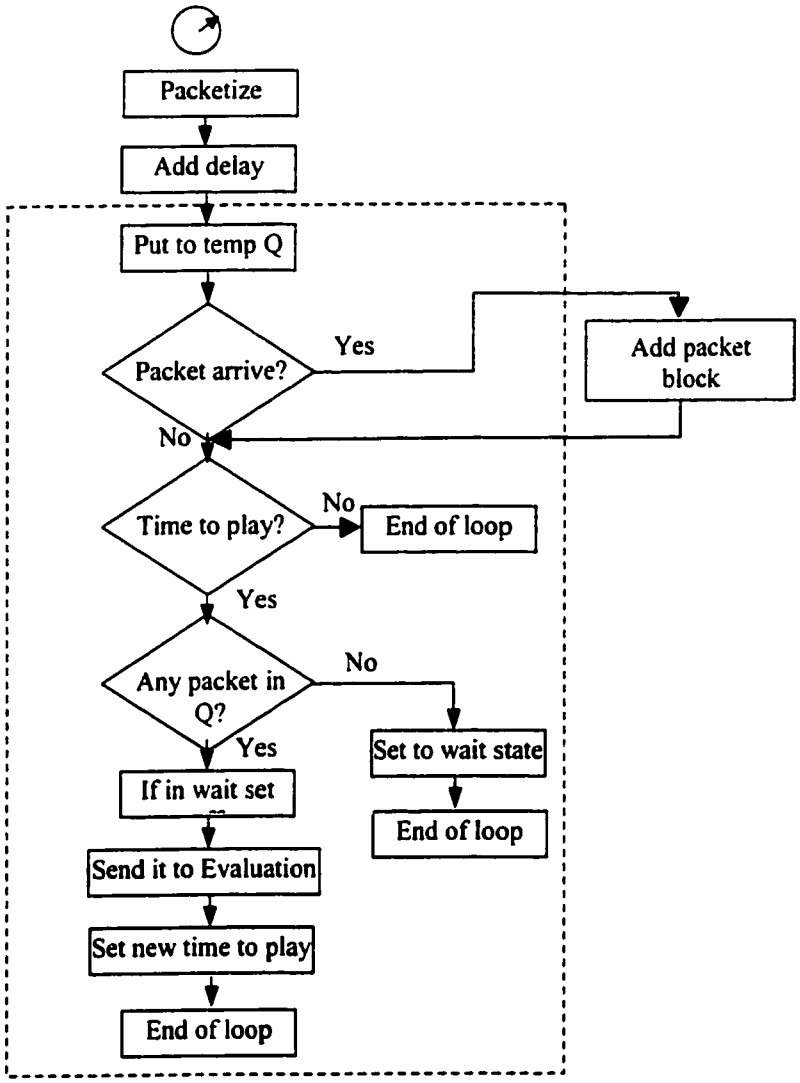


Figure 4.3 a) Diagram of jitter management

management block immediately. The jitter management block receives the packet and the delay generated for this packet. It is the jitter management block's responsibility to find when the packet arrives. The second queue is to save these packets temporarily. In

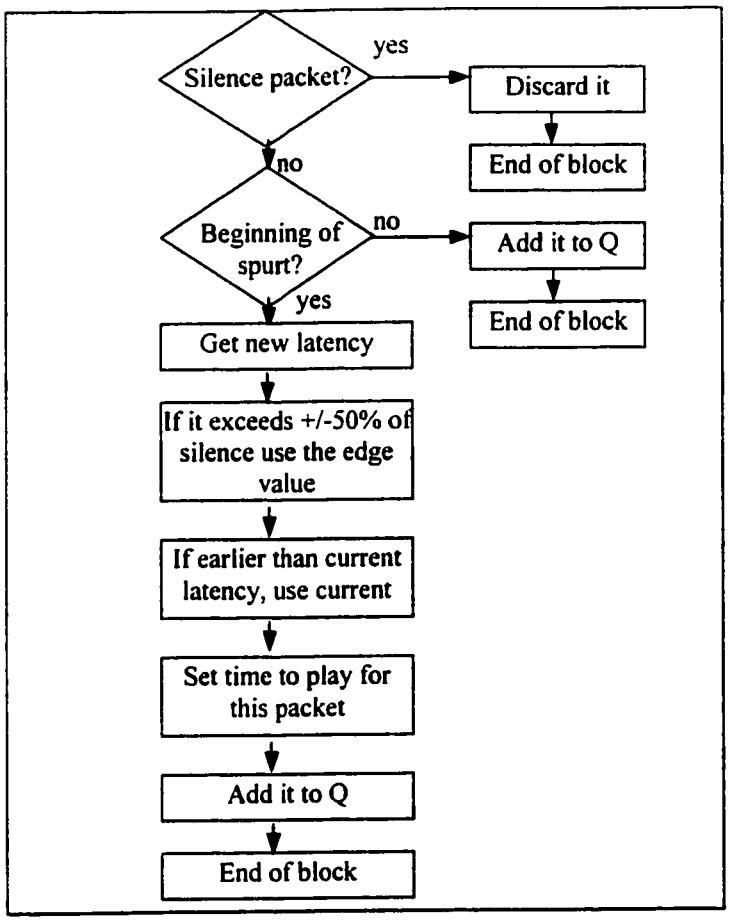


Figure 4.3 b) Add packet block

each loop, the packets in the temporary queue are checked. If current time = timestamp + delay, the packet is removed from the temporary queue, and passed to an 'add-packet component'.

In each loop, the display queue is checked to find if it is time to remove a packet. The system manages the queue by maintaining a global 'time-to-play'. So the decision of whether to remove a packet becomes a simple task of comparing the current time with the time-to-play. If they are equal and the display queue is not empty, then the oldest packet is removed from the queue and passed into the next block. Since normally the packets are played continuously, the time-to-play is easy to maintain. Every time we remove a packet, we just update the time-to-play with time-to-play + samples of a packet. But there

are two exceptions. One is at the beginning of a talk-spurt. (The talk-spurt is detected by silence detection algorithm, which will be covered in later sections.) The time-to-play for the first packet of a talk-spurt is decided by the add-packet component according to certain criteria. We will revisit it in the following discussion. The other exception is when the queue is empty. At this moment the system is set to waiting state. For the E-policy, which is shown in figure 4.3 a), during a waiting state the process doesn't check the time-to-play, but waits for a new packet. The system will be staying in the waiting state until a new packet comes, and then the packet is removed immediately from the queue into the next block, and the time-to-play returns to normal working state. For the I-policy, the situation is handled in a different way. In this case, we have to keep the same latency. So, once the current time is equal to the time-to-play and there are no packets in the display queue, we set the time-to-play to one frame time later, and discard the coming packet. As a result, a gap of a frame time is left in the playout. Because we assume there is no packet loss in the network, we don't check the sequence number of packets. For the threshold policy, in waiting state the system checks if any packet arrives before the threshold. If any packet arrives before threshold, the packet is played, and the time-to-play returns to its normal state. Once the threshold is approached, the system works in the same way as in the I policy. It should be pointed out that the latency is accumulated. So, we use an attribute to record how close the latency is to the threshold.

4.4.2 Latency Adjustment

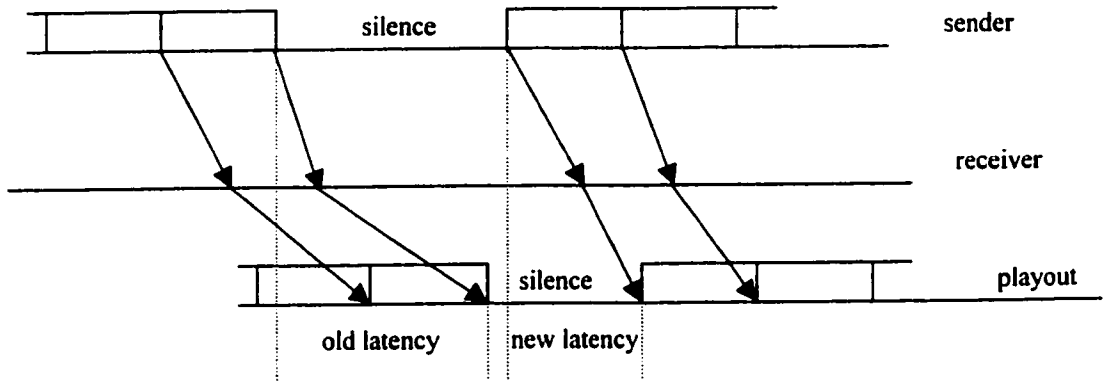
In the add-packet component (Figure 4.3 b), the delay of every incoming packet is recorded in a link list with length *m*. The link list is designed to be able to only record the

most recent m samples. The delay data in the list are also kept in a descending order, to facilitate the adaptive latency selection. The silent flag of the packet is checked. (Silence detection will be discussed in later sections.) If it is a silent packet, we only record its delay, and then discard this packet.

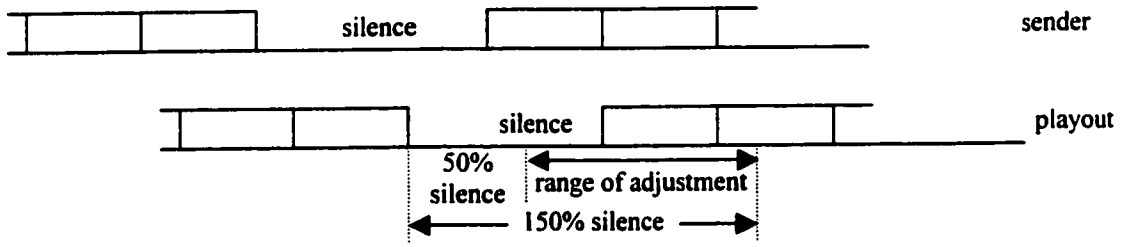
If the packet is the first packet of a talk-spurt, then it is the time to adjust the display latency. The recent delay history is checked, and the *k*th largest value is selected as the new display latency. k was also set in the configuration. Once the new latency was selected, we will check if it is applicable. In two situations, the new latency is not applicable. One situation is that the new latency changes too much from the old latency, either increasing or decreasing. The other is that first packet already arrived later than the new latency. For the latter situation, we have to use the current latency as the new latency. For the former situation, if the new latency is outside of the range for the change, we replace the selected latency with the boundary value.

Adjusting display latency will stretch or compress the silence period. The scenario is shown in figure 4.4. Generally speaking, the audio quality is robust to changing of silence periods, compared to active periods. But if the modification exceeds a threshold, it has impact on audio quality. Some silent periods, such as the periods between syllables of words, are quite short and sensitive to the modification. Experiments show that if the change is within 50 percent, it will not be perceptible [26]. The scenario shown in figure 4.4 a) is changing latency from high to low. The old latency is caused by high network traffic in a previous time. At the beginning of a talk-spurt, the system finds that the long buffering time is not necessary. So it decides to shorten the latency. But the action will result in the change of length of silent period. The range we set is between 50 percent and

150 percent of the original silent length (figure 4.4 b). If the latency is less than 50%, we use the low boundary value, and if the latency is more than 150 %, we use the high boundary value.



a) the impact of latency adjustment on silence period



b) range of latency adjustment

Figure 4.4 Adjust latency at the beginning of talk-spurt

After passing the two checks, the new latency is selected. As introduced before, the buffer is managed by maintaining a global 'time-to-play'. But we can't simply set the time-to-play to the calculated latency at this moment. The reason is that there may be still packets of the last talk-spurt in the queue. This occurs when the old buffering time is long and the silent period is short. In this case, we follow a logic to set the time-to-play (figure 4.5). If the packets for the last talk-spurt have finished with playing, we just set the time-to-play to the calculated latency. If not, we save the calculated latency, count the left packets, save the number of packets in a global attribute, and set the system into a special

system. In the buffer management process, once the system is set into this special state, it starts to count the packets. When it is turn for the first packet of the new talk-spurt, the time-to-play is set to the calculated latency saved in the add-packet process.

The first criteria in checking the applicability of latency belongs to E-policy. In I policy,

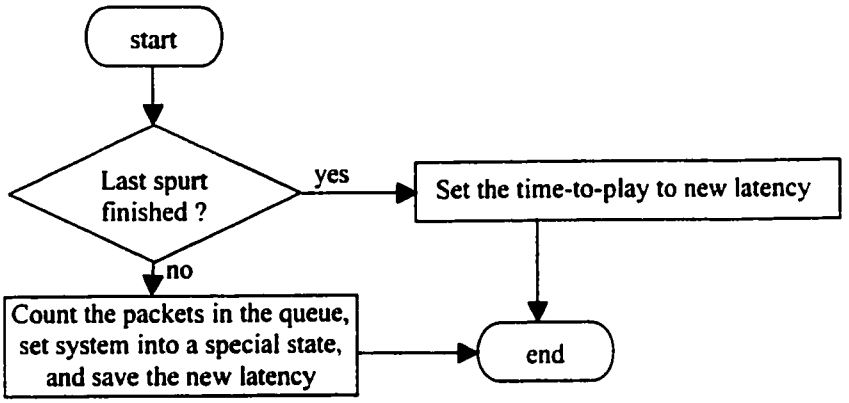


Figure 4.5 maintain the time-to-play

the new latency calculated from the delay history must be strictly followed. So if the first packet has arrived later than the new latency, it is discarded. For the threshold policy, if the first packet arrived before the new latency + threshold, it is kept and the latency is expanded, otherwise it is discarded as I policy. But the second criterion is the same for the I policy and the threshold policy.

4.5 Silence Detection and Hangover

4.5.1 Silence Detection Algorithm

In some packet voice applications, silence detection is used to use the bandwidth efficiently by not transmitting silence packets. But in this adaptive jitter management system, if we don't transmit the silence packet, we will loose the current network condition. So, here we use silence detection only to detect the talk-spurt.

Silence detection is applied in the packetizing block, and the result is marked in a field in the packet header. Since we change the latency at the beginning of a talk-spurt, the performance of jitter management largely depends on the performance of the silence detection algorithm.

The algorithm is based on [5], and modified to meet the requirement of this application. It is an adaptive algorithm that can give good performance under variable noise. The energy of a packet is represented by the average energy of the packet in dB.

$$E = 10\log(E_{avg} / E_{\max}) \qquad (4.1)$$

The E_{avg} and $E_{\max}$ are the energy expressed in form of square of amplitude. We set the dB range from 0 to -60, and then translate the energy to a relative dB ranging from 0 to 1.

$$RE = 1 + E/60 \qquad (4.2)$$

In the implementation, a lookup table is used to transform the PCM data to relative dB.

We set a maximum threshold T_{max} and a minimum threshold T_{min}, and the current threshold can only vary between the two values. The algorithm maintains an average noise energy, and sets the threshold as the average noise plus a small constant energy.

A default threshold is set at the beginning. For each incoming sample, its value is compared with the current threshold value. If the incoming value is larger than the threshold, then the algorithm concludes that it is an active packet. The threshold is increased by a small step. We will discuss later that this increase is necessary. If it is less than the threshold, the algorithm concludes this packet is noise, and it is returned as a silence packet. At the same time, this value of this packet is used to update the average noise energy and the threshold.

$$E_{avgnoise} = E_{avgnoise} + ratio \times (E_{noise} - E_{avgnoise}) \quad (4.3)$$

$$Threshold = E_{avgnoise} + E_{constant} \quad (4.4)$$

All the values here are converted in dB. The new threshold is compared with the maximum threshold and the minimum threshold. If it is larger than the maximum one, the threshold is set to maximum value, and if it is smaller than the minimum value, the threshold is set to the minimum value. The diagram is shown in figure 4.6

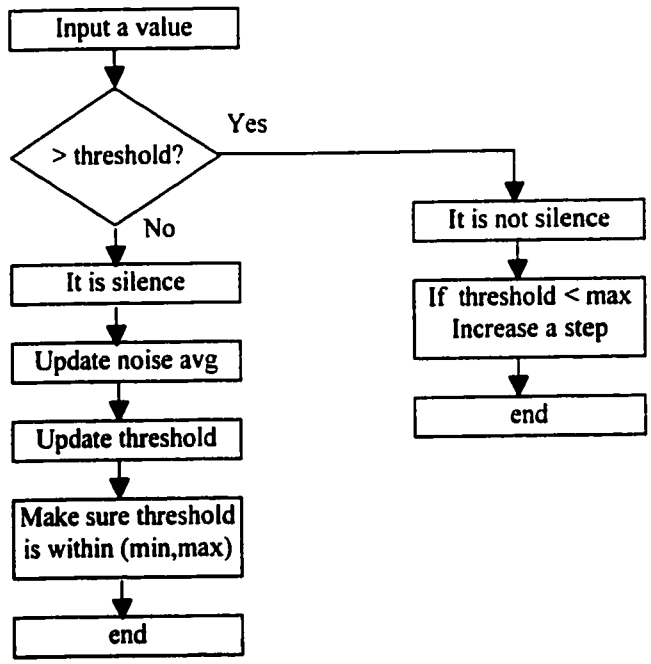


Figure 4.6 Diagram of silence detection

There are some flexible parameters in this algorithm. How to select the values for these parameters determines the performance of this algorithm. Let us see how the algorithm works when the noise level increases and decreases.

Looking at figure 4.7, from packet 2, the energy of packets starts to be lower than the current average noise energy, which represents a decrease of noise level. From equation 4.3 and 4.4, we understand the average noise energy and threshold will decrease to adapt the change. This adaptive strategy will make the algorithm to be able to detect a low

energy signal. Looking at packet 5, we find it is detected as active packet with this adaptive algorithm. While in a fixed threshold approach, it will still be considered as silence packet, which is not appropriate. The average noise energy actually is a weighted average value of recent noise, with the more recent sample the more weight. The ratio in equation 4.3 determines the changing speed of noise energy. A too large ratio causes the system too sensitive to noise change and unstable. A too small ratio will give the old samples too much weight and make the system slow to noise change. We must make a compromise between the system stability and response speed. In this application we choose the ratio at 0.1.

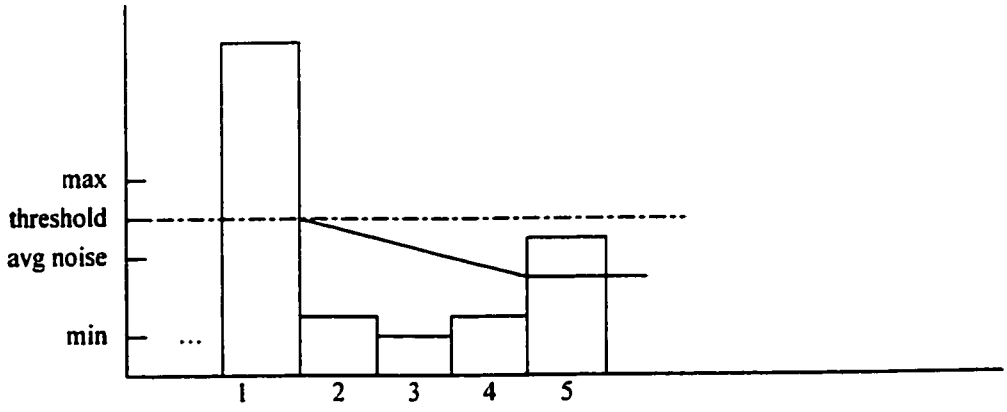


Figure 4.7 silence detection algorithm adaptive to noise decrease

The situation is more complicated when the noise increases. We have to consider two possibilities. One is that the noise increases, but the energy of the coming noise is still under the current threshold. In this case the strategy that can deal with the decrease of noise can still deal with this kind of change. But what will happen when the noise increases, and the energy of the noise packets is a little bit higher than the current threshold? (Figure 4.8) If the energy is still lower than the maximum threshold, the

current threshold should be increased. The above strategy will fail to respond to this kind of change. Without other strategy, the threshold will not be adaptive in this situation. So we introduce another policy for this case. Whenever a packet is recognized as active packet, we increase the threshold by a small step. So if the mentioned change happens,

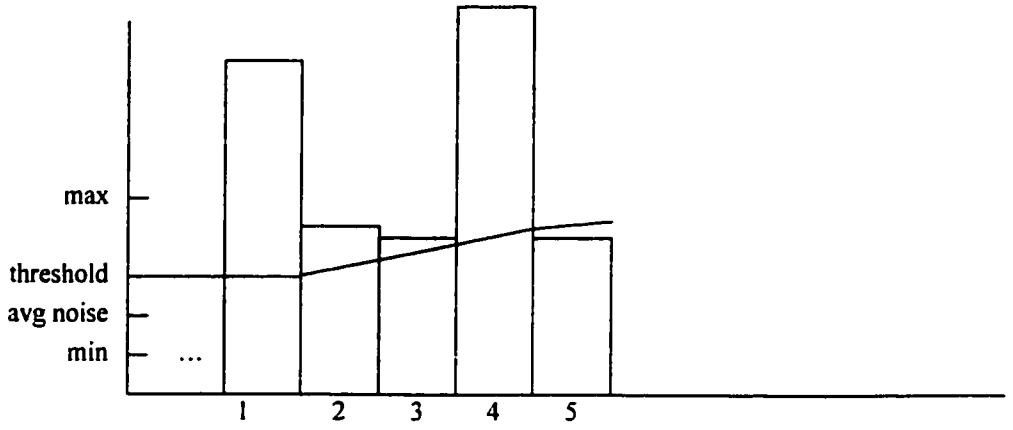


Figure 4.8 silence detection algorithm adaptive to noise increase

the threshold will approach the maximum threshold slowly. After a while the current threshold will be greater than the new noise energy, and then the new coming noise packets can be used to update the average noise energy. In figure 4.8, the noise starts to increase from packet 2. The first several packets are still treated as active packets. But because the threshold increases a step on every active packet, from packet 5 the coming noise packets will be treated as noise and used to update the average noise energy. In fact the step is quite small. In our application, we set the step as 0.06 dB (in relative dB). The side effect of this strategy is that it causes the threshold to vibrate. But because the vibration is within a small range, we would like to make this compromise.

4.5.2 Hangover Strategy

We find that, with silence detection, there are a lot of short silence periods in an audio stream. The purpose of using silence detection is that, in our opinion, the audio is robust to changes of the length of the silence periods. But short silence is not the silence we would like to detect, because it represents an interval inside a word (not between words), and stretching or compressing this kind of silence will have impact on the quality of voice. So we use a hangover strategy as shown in figure 4.9. We append a fixed length of period to each talk-spurt, so as to merge those silence periods with length shorter than the length of hangover. In this way we create longer talk-spurts.

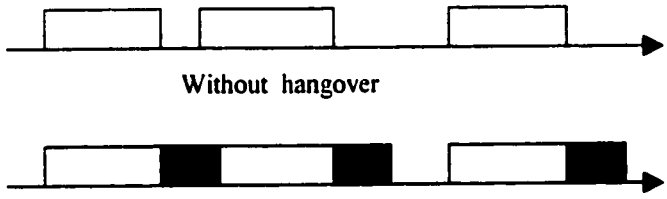


Figure 4.9. Hangover strategy

4.6 Traffic Generator

A traffic generator is used to generate a delay sequence to simulate end-to-end network delay variance. As mentioned before, there is not a perfect model for end-to-end delay. In some theoretical analysis, an independent exponential distribution is used to generate network delay jitter. Although the observed network delay has a distribution similar to an exponential one, but we can't generate a delay sequence from a given distribution. When generating exponential distributed data, we usually randomly generate a probability between 0 and 1, and then calculate its corespondent value from the following

equation: $F(x) = 1 - e^{-\mu x}$. The problem is the generated data are independent. Let the delay of packet i be d(i), and the delay of packet i+1 be d(i+1). Assume the packet time is T. Then it is possible that d(i) – d(i+1) > T, which means the older packet arrives earlier than the new packet. In this simulation, we assume that all the packets are transmitted through the same path. So we don't allow out of order packets. Although we can avoid out of order packets by limiting the maximum delay (< packet time), this limitation prevents the generator of generating heavy traffic. In this application we give an option to generate an independent exponential distribution delay. We limit the maximum delay to no more than a frame time. But we recommend using the following end-to-end delay model.

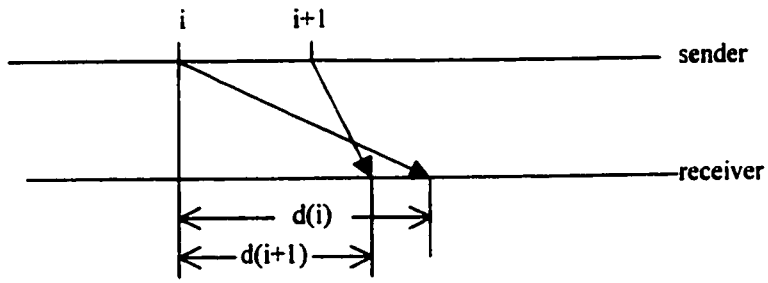


Figure 4.10 When d(i) larger than d(i+1)

The end-to-end model used in this application was proposed in [37] to study Internet packet loss and delay behavior. It is a simple single-server queuing model with two input streams, where one stream represents the audio traffic and the other stream represents the background traffic (figure 4.11). The audio stream has constant arrival and constant service time. There is a constant delay for the audio. The background traffic stream has Poisson arrival and exponential service time stream. Since we don't consider loss of packets, we assume an infinite buffer size. With this model, we can adjust parameters to generate different jitter sequences. First the above problem is solved. Since the audio

packets and Internet packets go through the same pipe, there is a correlation between consecutive packets. This model doesn't generate out of order packets. By changing the arrival rate of the audio stream, we can simulate audio streams with different frame times. By adjusting D, we can change the fixed part in the end-to-end delay. By varying the average arrival rate and average service time of the internet data stream, we can insert high and low background traffic. A comparison of the result generated from this model and experiments from a real network shows that the statistical behavior of this model is

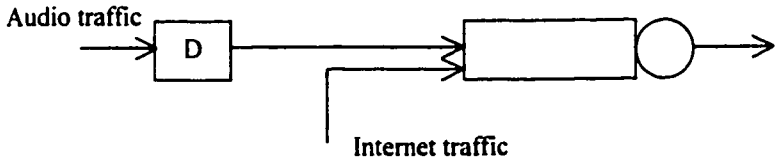


Figure 4.11. A model for end-to-end delay

similar to that of a real network. In this model, we bound the maximum delay by bounding the maximum arrival rate of the internet traffic stream.

The application offers two options to add delay. One is using an exponential delay generator to generate delay on the fly. The other one is reading from a file. Using the above end-to-end model, we can record its output of delay sequence into a text file. We can specify the file name in the configuration GUI of the application, and then the application reads the delay in order and adds it to the packets. With the second option, users can also use other delay models to generate delay files to be used in this application. A good point is to record the delay in a real network, and use it as input in this application. It will give more practical test to the jitter management algorithms.

4.7 Evaluation Block

Generally speaking, audio evaluation is quite application dependent, and there is no standard. Quality of audio can be evaluated by objective methods and subjective methods. Subjective methods means asking people to hear the audio and give marks to the quality perceived. The quality is given from the statistical results. Objective methods means using a metric, usually a set of parameters, to represent the quality of audio. It is considered that subjective methods can give more reasonable evaluation. But objective methods are easy to implement and comparable. In this application, we use a simple objective evaluation method, and at the same time we record the actual output audio.

The output of the jitter management block is non-silent audio packets with three timestamps for each packet. The timestamps are the sending time, receiving time, and playout time. The first function of this block is to record the audio stream just as it is played in an output device. The system checks the play time for each packet. By comparing play time of consecutive packets, it finds the gaps between packets. We didn't apply any recovery mechanism in this application, but only filled the gaps with blanks. We copy the file header from the input wave file and write it with the stream data into another wave file.

In this application, we use average delay and gap frequency, expressed in gaps per minute, to evaluate the performance of jitter management. Because we use much longer packet length (20 ms) in the experiments, we add an average gap length and total gap length as extra parameters to evaluate. The gap frequency is represented in number of gaps per minute. This evaluation method can generally describe the performance of a jitter management algorithm. But it is very simple, and there are some features that can

not be described by this method, such as the burst of the gaps. The other problem is, with this method, the measurements are not always comparable. For example, algorithm A has longer average delay and smaller gap frequency than algorithm B under the same traffic. To avoid this problem, we only compare the measurements that are comparable. If algorithm A has almost the same average delay with algorithm B, but its gap frequency is higher than algorithm B, we conclude that algorithm A has better performance.

4.8 Experiment Design and Results

We design a set of experiments to test the performance of the jitter management algorithms and also to test the simulation tool. In the experiments, we first generate two delay files with the end-to-end delay model, one representing light traffic, and the other representing heavy traffic. For the light traffic, we have the average delay at 2.79 ms, standard deviation at 1.54 ms, and maximum delay at 21 ms. For the heavy traffic, we have average delay at 10.49 ms, standard deviation at 8.9 ms and maximum delay at 55 ms. The audio input is a 2 minutes wave file. It is a mono channel, 16 bits, 8000 Hz sampling rate speech audio. The packet size is 320 octets, which represent 20 ms of audio. The hangover in the silence detection algorithm is two packets long, which is 40 ms. The parameters in the silence detection are set as follows.

Max threshold = - 20 dB;

Min threshold = - 45 dB;

Initial threshold = max threshold;

Initial average noise = -24 dB;

increase step = 0.06 (relative dB);

threshold above noise = 4 dB;

For light traffic and heavy traffic, E policy, I policy, and threshold policy are applied. For E policy and I policy, we vary m and k/m ratio, and collect the average delay, gap frequency, and average gap. For the threshold policy, we select a fixed m/k ratio, very the threshold, and collect the evaluation data. All the audio outputs are also recorded as a wave file.

Group 1

Gap frequency and average delay versus m with k/m at 0 and 10% for E policy and I policy under low traffic.

	k/m=10%		k/m=0%	
m	gap freq	avg gap	gap freq	avg gap
10	109.5	2.37	79	2.3
20	100	2.4	50.5	2.23
40	92.5	2.41	31.5	2.17
60	90	2.47	23	2.11
80	89	2.44	19.5	2.03
120	87.5	2.46	16	1.88

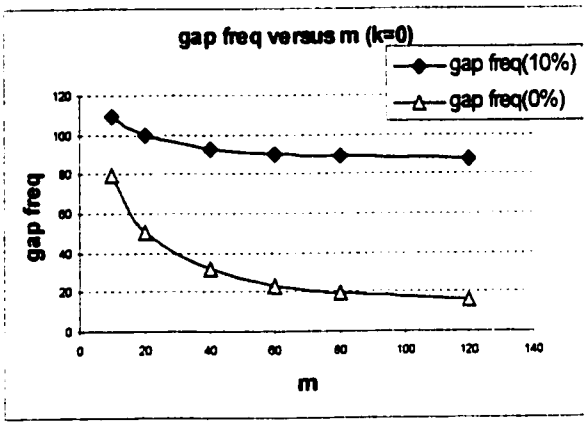


Figure 4.12. Gap frequency of E-policy with m(10-120) k/m=0 and k/m=10% in low traffic,

m	Delay(10%)	delay(0%)
10	7.02	7.75
20	7.01	8.34
40	7.05	8.98
60	7.04	9.57
80	7.06	9.95
120	7.07	10.57

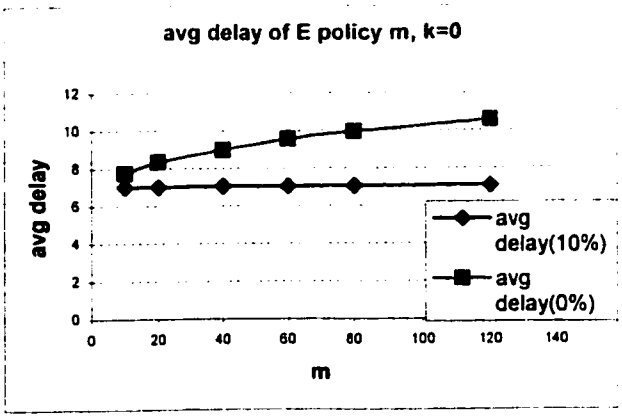


Figure 4.13 Average delay of E-policy m(10-120) k/m=0 and k/m=10% in low traffic.

	k/m=10%		k/m=0%	
m	Gap freq	avg gap	gap freq	avg gap
10	253	25.34	158	24
20	212	24.29	74	23.51
40	177.5	24	40	23.5
60	178.5	23.6	29.5	23.39
80	168.5	23.8	23	23.04
120	163.5	23.9	17	22.35

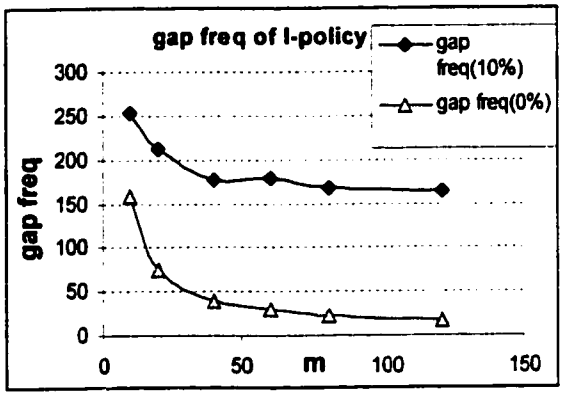


Figure 4.14. Gap frequency of I-policy m(10-120) k/m=0 and k/m=10% in low traffic,

m	avgdelay(10%)	avg delay(0%)
10	4.18	6.06
20	4.33	7.37
40	4.49	8.37
60	4.46	8.98
80	4.56	9.53
120	4.66	10.26

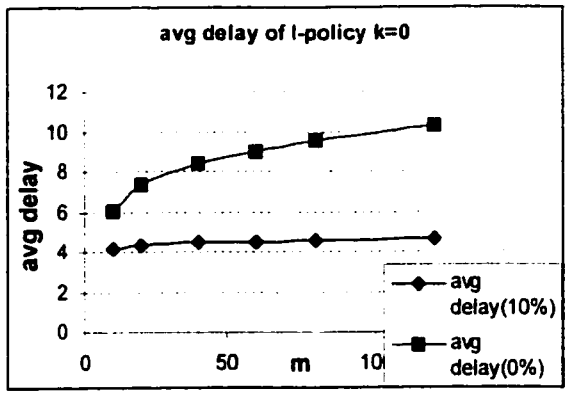
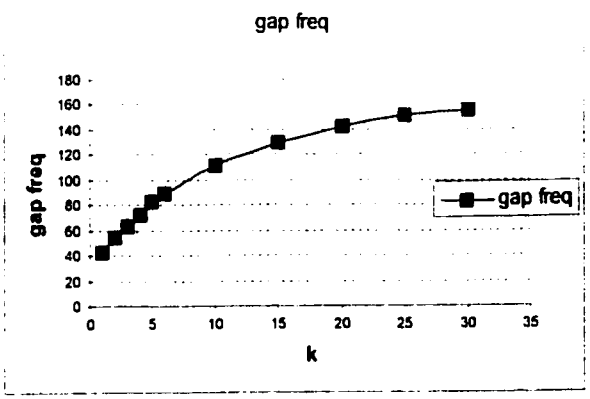


Figure 4.15. Average delay of I-policy m(10-120) k/m=0 and k/m=10% in low traffic,.

Group 2

Gap frequency and average delay versus k with fixed m at 60 for E policy and I policy under light traffic.

k	avg delay	gap freq	avg gap
1	8.3	43	2.36
2	7.81	55	2.48
3	7.45	63	2.56
4	7.26	72	2.53
5	7.15	82.5	2.45
6	7.04	90	2.47
10	6.88	111	2.42
15	6.79	130	2.35
20	6.73	142.5	2.38
25	6.72	151.5	2.36



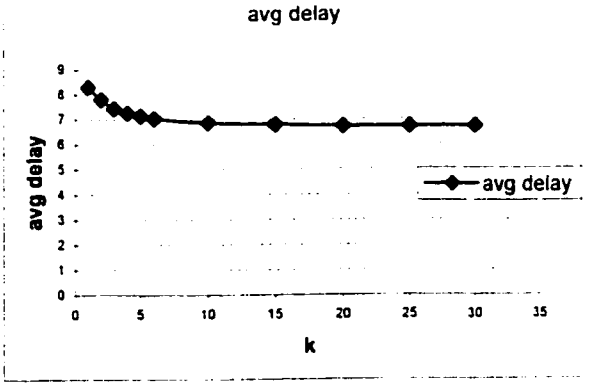
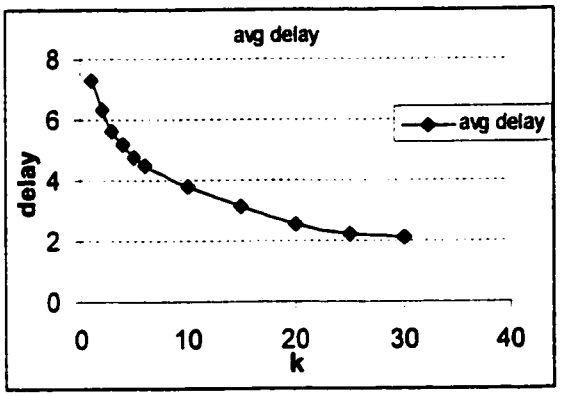


Figure 4.16. Average delay and gap frequency

of E-policy m:60, k:0-30 in low traffic,

k	avg delay	gap freq	avg gap
1	7.32	57	23.51
2	6.32	76.5	23.53
3	5.6	98	23.27
4	5.2	121	23.3
5	4.78	149	23.89
6	4.46	178.5	23.6
10	3.79	252	24.6
15	3.13	344	25.9
20	2.52	422.5	28.54
25	2.22	472.5	29.88
30	2.09	487	30.37



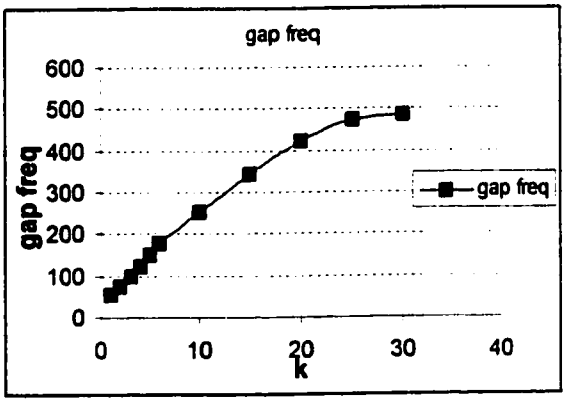
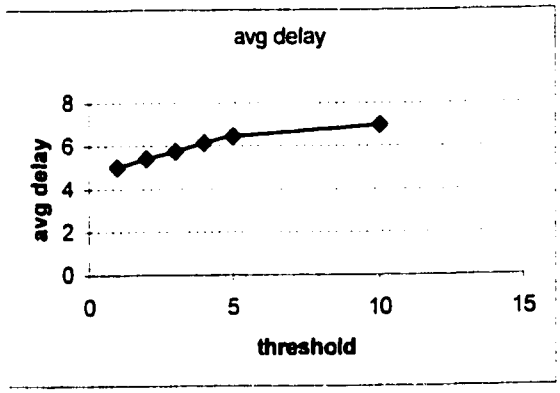


Figure 4.17. Average delay and gap frequency of I-policy m=60, k=0-30 in low traffic,

Group 3

Average delay and gap frequency (or total gap length) versus threshold for threshold policy under light traffic.

T	avg delay	gap freq	avg gap	freq*length
1	5.02	138.5	16.9	2340.65
2	5.41	120.5	11.7	1409.85
3	5.75	114	9.07	1033.98
4	6.14	101.5	6.48	657.72
5	6.45	95	4.68	444.6
10	6.98	89.5	2.54	227.33



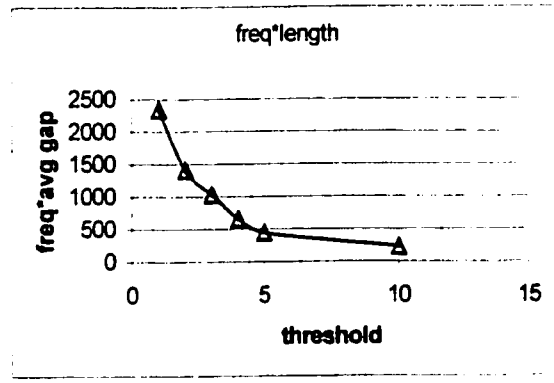


Figure 4.18.Average delay and total gaps, threshold policy, m=80 k=8 t=0-10 in low traffic.

T	avg delay	gap freq	avg gap	freq*length
0	3.51	290.5	25.47	7399.035
1	4.13	223	18.99	4234.77
2	4.69	183.5	13.9	2550.65
3	5.11	159.5	10.1	1610.95
4	5.46	148.5	7.78	1155.33
5	5.85	136.5	5.79	790.335
10	6.75	120	2.58	309.6
E	6.86	119.5	2.39	285.605

Figure 4.19. Average delay and gap frequency of threshold in (80,16) t=0-10 in low traffic

Group 4

Gap frequency and average delay versus m with k/m at 0 and 10% for E policy and I policy under low traffic.

	k/m:10%		k/m:0%	
m	gap freq	Avg gap	Gap freq	avg gap
10	154	4.29	125	4.27
20	140	4.16	93.5	4.24
40	118	4.1	61.5	4.58
60	104	4.07	48	4.28
80	93.5	4.2	40	4.275
120	90.5	4.27	34	4.04

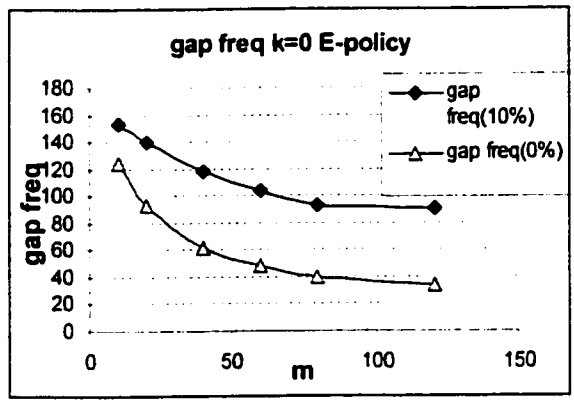


Figure 4.20. Gap frequency of E-policy k/m=0 and k/m=10% in high traffic.

m	delay(10%)	delay(0%)
10	24.2	25.23
20	24.78	26.81
40	25.99	29.76
60	26.8	32.78
80	27.03	33.8
120	26.81	35.3

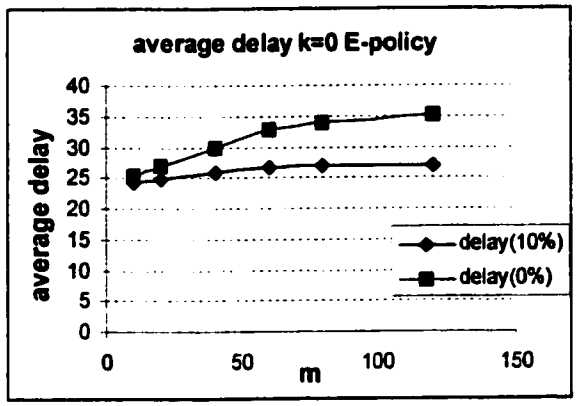


Figure 4.21. Average delay of E-policy k/m=0 and k/m=10% in high traffic.

	k/m=10%		k/m=0%	
m	Gap freq	Avg gap	Gap freq	avg gap
10	135.5	61.85	99	65.05
20	123.5	61.78	77	61
40	102.5	55.51	48	57.5
60	83	59.88	36.5	54.2
80	75.5	61.19	30.5	61.3
120	68.5	63.8	23.5	66.38

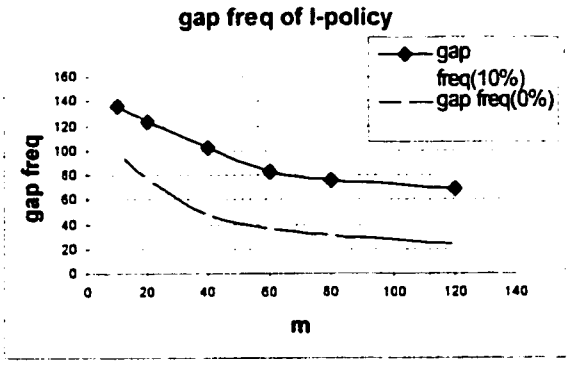


Figure 4.22. Gap frequency of I-policy k/m=0 and k/m=10% in high traffic.

m	delay(10%)	delay(0%)
10	17.55	20.57
20	18.48	23.08
40	20.97	27.33
60	22.46	31.27
80	22.63	32.54
120	22.38	34.2

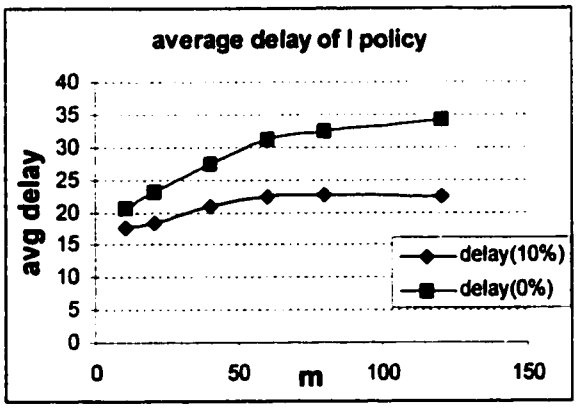
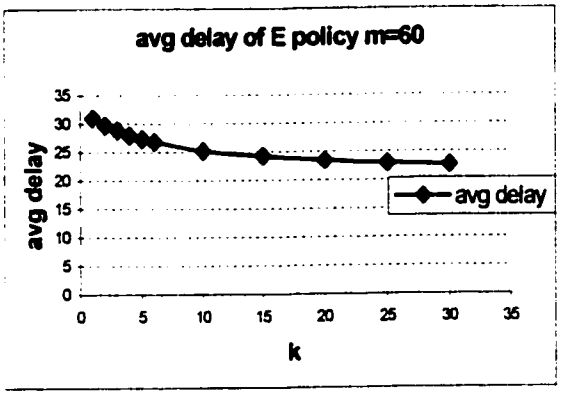


Figure 4.23. Average delay of I-policy k/m=0 and k/m=10% in high traffic.

Group 5

Gap frequency and average delay versus k with fixed m at 60 for E policy and I policy under heavy traffic.

k	avg delay	gap freq	avg gap
1	30.97	62	4.27
2	29.7	71.5	4.27
3	28.87	78.5	4.29
4	27.95	85	4.34
5	27.34	95.5	4.2
6	26.8	104	4.07
10	25.2	128	4.16
15	24.22	154	4.22
20	23.43	180.5	4.21
25	22.94	200.5	4.23
30	22.61	214.5	4.28



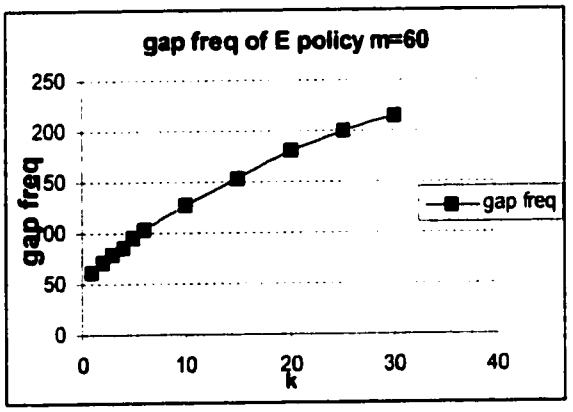
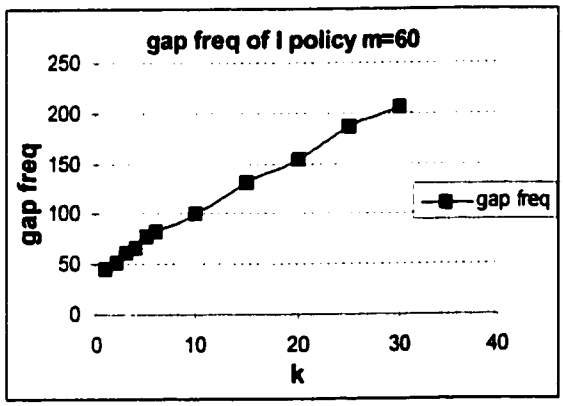


Figure 4.24. Average delay and gap frequency of E-policy m=60, k=0-30 in high traffic,

k	avg delay	gap freq	avg gap
1	28.74	45.5	62.86
2	27.06	52	63.65
3	25.61	61.5	55.45
4	24.39	66	60.3
5	23.31	78	58.33
6	22.46	83	59.88
10	19.25	100.5	63.68
15	16.54	131	65.95
20	13.8	154.5	69.71
25	11.5	186	72.47
30	9.5	206	72.67



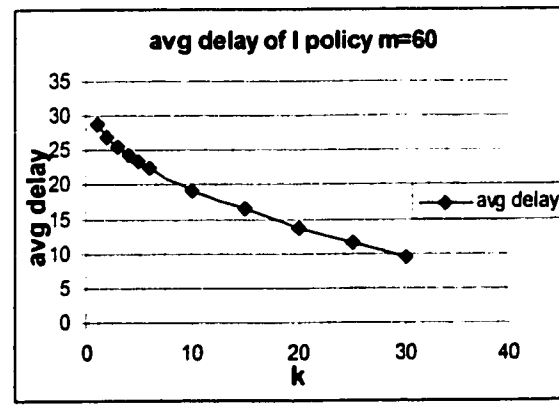


Figure 4.25. Average delay and gap frequency of I-policy m=60, k=0-30 in high traffic.

Group 6

Average delay and gap frequency (or total gap length) versus threshold for threshold policy under heavy traffic.

T	avg delay	gap freq	avg gap	total gaps
0	20.97	102.5	55.51	5689.775
5	22.44	104.5	32.85	3432.825
10	23.87	114.5	17	1946.5
15	24.78	113	11.26	1272.38
20	25.34	118	6.96	821.28

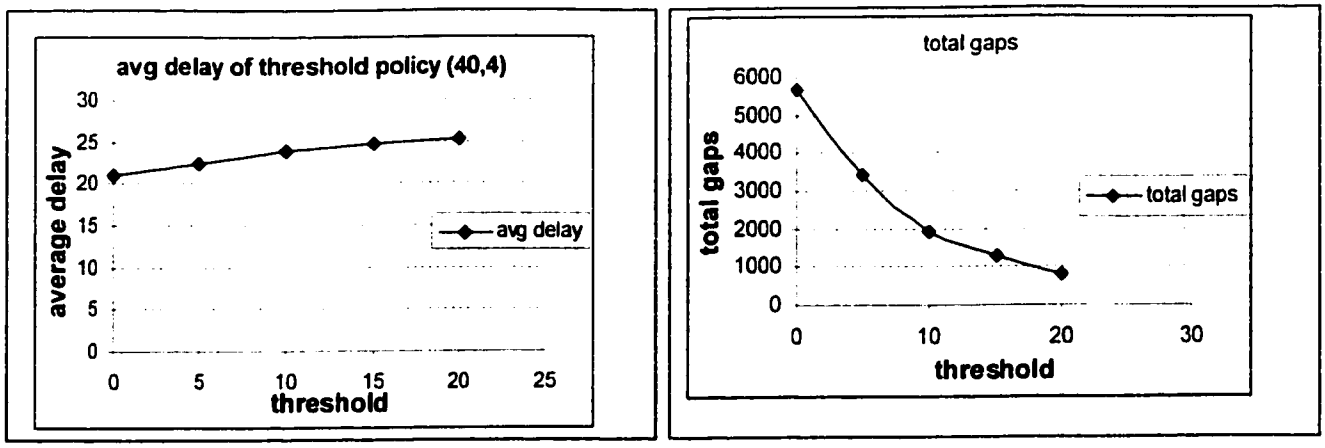


Figure 4.26.Average delay gap frequency of threshold policy (40 ,4), t=0-20 in high traffic,

T	avg delay	gap freq	avg gap	
0	18.11	122	62	7564
5	19.72	132.5	33.2	4399
10	21.53	130	23.06	2997.8
15	22.87	131	13.88	1818.28
20	23.78	135.5	8.42	1140.91
30	24.58	138	4.68	645.84
e	24.76	136	4.18	568.48

Figure 4.27.Average delay and gap frequency of threshold policy(80,16) t=0-30,high traffic

4.9 Analysis and Conclusions

It has been discussed in last chapter that choosing appropriate history length (m) is very important. A too small m doesn't give us the correct estimate of the current network situation. While a too large m will consume too much computer resources and at the same time it is not necessary. From results of group 1 and group 3, we find, as expected, for both high traffic and low traffic, the quality increases with m until m is around 50, from where the average delay and gap frequency don't change too much. This tells us that by

using a history length around 50, we can have approximate knowledge of the current network.

Groups 2 and 5, show how quality changes with k when m is at fixed length of 60 under light traffic and heavy traffic. We noticed that for the E policy, the average delay decreases with increasing k (which means choosing smaller display latency). But when k is larger than 10 (where k/m is 17%), the delay doesn't improve much, while the gap frequency still increases. The explanation for this phenomenon is given below. The

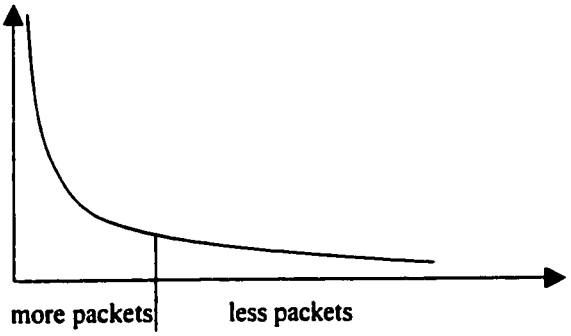


Figure 4.28 less packets with large delay

distribution of delay is similar to exponential distribution with most delay at low level and fewer packets at high level. The E-policy expands the latency whenever meeting a longer delay. When the display latency is chosen at a rather small value, it doesn't help much to decrease the average delay. The reason is that there is a large chance that the latency will be expanded in the first few packets. So it is good to keep k less than 17% of m for E policy.

For the I policy, it is different. Let's compare the E policy with the I policy. It should be mentioned that the packet size used in this experiment is 20 ms. So discarding a packet will cause a 20 ms gap. From figure 4.17 and figure 4.25, we find, in light traffic the average gap is close to packet length, while in heavy traffic the average gap is up to several times of packet length. The end-to-end delay model generates many consecutive

long delays when running at heavy traffic (which is similar to the practical situation). Having this in mind, we understand that the gap frequency of the E policy and the I policy are not comparable. By hearing the audio output of the I policy, we find its intelligibility is much poorer than the output from the E-policy with the same or less gap frequency. The audio quality is very sensitive to packet loss, especially when the packet size is large or the packet loss is consecutive. So unless the time requirement is very strict, we should avoid using the I policy. In figure 4.25, by setting k/m close to 50% (which means we will loss half of the packets), we can have the average delay close to the average network delay. But the quality is already quite poor, with the average gap at 70 ms and gap frequency at 200 per minute.

The threshold policy gives us a compromise between E policy and I policy. Figure 4.18 shows the threshold policy running at m=80 and k=8. Figure 4.17 shows the I policy with m=60 and k=0-30. Comparing these two figures, we see the threshold policy at threshold 2 gives the same average delay as the I policy at m=60 and k=4, but only has half gap length and the same gap frequency. The high traffic situation shows a similar result. We understand that increasing the threshold from 0 to infinity actually goes from the I policy to the E policy. So the average gap length increases sharply with the increasing of the threshold. It is difficult to compare the threshold policy and E policy. We think the threshold policy can be used in such a situation, that the play latency requirement is very strict. We use threshold policy with a large k/m ratio and a small threshold. The small threshold is used to control the length of gap and gap frequency. It is difficult to choose such an appropriate threshold. A possible solution is to use the current average gap length or packet loss rate as feedback to dynamically adjust the threshold.

Chapter 5 Conclusion

In this thesis, we have first surveyed speech coding technologies. These technologies provide us with a wide selection from high bit rate wave form codecs (i.e. PCM) to low bit rate codes (i.e. LPC). We have also surveyed packet switched network technologies. Today's LAN technologies are characterized by media sharing. In a LAN environment, all the nodes share a LAN or part of a LAN (segment), and a media sharing protocol is provided to make sure the medium is shared fairly. In a WAN area, the channels are shared statistically, and the packets are routed independently in a store and forward mode. Generally speaking, the current data network doesn't provide functionality to transfer real-time traffic.

Although, eventually, providing real-time service over packet switched networks relies on modifying network structure, application adaptation gives us an alternative to building real-time applications over current networks. Application adaptation means that, instead of changing the network itself, we apply control strategies only at the application layer to compensate the network problems. Bandwidth control is a strategy to vary the bit rate according to the network congestion. It is proved to be a perfect solution in a controlled LAN environment, but fails when applying to an uncontrolled WAN environment. Packet loss recovery strategies can efficiently improve audio quality. Among different recovery algorithms, the redundancy algorithm has the best performance, and is adaptive to network conditions. Network delay variance is considered to be the key problem to carrying real-time traffic over packet networks. The basic idea of delay jitter compensation is buffering the audio to smooth the jitter. In buffer management, there is a

tradeoff between display latency and gap frequency. It is considered that low latency with acceptable gap frequency is better than high latency without gaps. Successful jitter management algorithms make a balance between these two factors and adjust the latency dynamically according to the network condition.

We have developed a simulation tool to implement and evaluate some typical jitter management algorithms. Wave files are used as the audio input to the tool. An adaptive silence detection algorithm is used to detect the talk-spurts in the audio stream. A network end-to-end model is used to generate delay jitter. A simple evaluation method is used to evaluate the performance of the algorithms. The algorithms implemented in this tool are flexible. They dynamically adjust the display latency at the beginnings of talk-spurts by observing the recent delay of packets. There are three different ways to deal with late packets, whether discarding (I policy), or waiting forever (E policy), or waiting within a threshold (threshold policy). We have studied the performance of the algorithms by varying the observed history length, the way of selecting latency, and the way to deal with late packets. The results show that with the packet length of 20 ms, a combination of m (history) around 50 and k/m (ratio) around 17% for E policy can give good performance. We should avoid using the I policy unless the time requirement is very strict. We anticipate that by using the average gap length or packet loss rate as feedback, the threshold policy with a large k/m and an appropriate threshold may help when the delay requirement is very strict.

The simulation study can be extended in the following ways. First, new end-to-end delay models can be introduced to evaluate the algorithms, especially the models that can generate burst traffic. It is also valuable to record the delay from a real network in

different times (peak time and idle time) and use it as input to this system. Another useful expansion is introducing clock offset and frequency difference between the sender and the receiver. Then without a clock synchronization mechanism, implement buffer management algorithms like 'queue monitoring' or 'holding time' to study its performance in such a situation, and design a strategy to deal with the problem. Further study can include simulation of packet loss, operating scheduling variance, and bandwidth variance.

References

[1] M. H. Chan and J. P. Princen, Prioritized Statistical Multiplexing of PCM Sources, IEEE/ACM Transactions on Networking, pages:549-559, VOL. 3, NO 5, October 1995.

[2] M. Borden, E. Crawley, and J. Krawczyk, Issues for RSVP and Integrated Services over ATM, Internet Draft, February 22, 1996.

[3] S. Deering, Host Extensions for IP Multicasting, RFC 1112, August 1989.

[4] H. Schulzrinne, RTP Profile for Audio and Video Conferences with Minimal Control, RFC 1890, January 1996.

[5] H. Schulzrinne, Guide to NeVoT 3.33, Technical Report, October 20, 1995.

[6] H. Schulzrinne, RTP: A Transport Protocol for Real-time Applications, RFC 1889, January 1996.

[7] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, Resource Reservation Protocol (RSVP) Version 1 Function Specification, Internet Draft, August 12, 1996.

[8] D. R. Boggs, J. C. Mogul, and C. A. Kent, Measured Capacity of an Ethernet: Myths and Reality, Technical Report, Western Research Laboratory, September, 1988.

[9] T. Bially, A. J. Mclaughlin, and C. J. Weinstein, Voice Communication in Integrated Digital Voice and Data Networks, IEEE Transactions on Communications, VOL. COM-28, NO. 9, pages:1478-1490, September 1980.

[10] S. R. Ahuja and K. G. Murti, Packet Telephony, Bell Labs Technical Journal, Spring, pages:5-14, 1997.

[11] T. Suda and T. T. Bradley, Packetized Voice/Data Integrated Transmission on a Token Passing Ring Local Area Network, IEEE Transactions on Communications, VOL. 37, NO. 3, pages:238-244, March 1989.

[12] S. Berson, IP Integrated Service Support in ATM, Internet Draft, June 1996.

[13] G. J. Coviello, Comparative Discussion of Circuit- vs. Packet-Switched Voice, IEEE Transactions on Communications, VOL. COM-27, NO. 8, pages:1153-1160, August 1979.

[14] V. Hardman and M. Handley, Reliable Audio for Use over the Internet, Technical Report, UCL, 1995.

[15] H. Saito, Optimal Control of Variable Rate Coding in Integrated Voice/Data Packet Networks, Performance Evaluation, VOL. 10, pages:115-128, 1989.

[16] W. Chen, A. Li, and M. Schwartz, A New Voice Scheduling Scheme for Broadcast Bus Local Area Networks, CH2702-9/89/0000/0448 IEEE, pages:448-457, 1989.

[17] R. Yavatkar, P. Pai, and R. Finkel, A Reservation-Based CSMA Protocol for Integrated Manufacturing Networks, IEEE Transactions on Systems, Man, and Cybernetics, VOL. 24, NO. 8, pages:1247-1258, August 1994.

[18] C. Szabo, An Ethernet Compatible Protocol to Support Real Time Traffic and Multimedia Applications, Computer Networks and ISDN System, VOL. 29, pages:335-342, 1997.

[19] F. A. Tobagi and N. G. Cawley, On CSMA/CD Local Networks and Voice Communication, Technical Report, CH1745-9/82/0000/0122 IEEE, pages:122-127, 1982.

[20] D. H. Johnson and G. C. O'Leary, A Local Access Network for Packetized Digital Voice Communication, IEEE Transactions on Communications, VOL. COM-29, NO. 5, pages:679-688, May 1981.

[21] T. A. Gonsalves, Packet-Voice Communication on an Ethernet Local Computer Network, ACM 0-89791-089-3/83/0300-0178, pages:178-185, 1983.

[22] F. Jia and B. Mukherjee, The Superchannel Scheme for Integrated Services on Multiple Access Broadcast Networks, Computer Networks and ISDN Systems, VOL. 27, pages:1523-1543, 1995.

[23] J. G. Gruber, A Comparison of Measured and Calculated Speech Temporal Parameters Relevant to Speech Activity Detection, IEEE Transactions on Communications, VOL. COM-30, NO. 4, pages:728-738, 1982.

[24] V. Hardman and M. Iken, Enhanced Reality Audio in Interactive Networked Environments, Technical Report, UCL, 1995.

[25] D. Ferrari, Delay Jitter Control Scheme for Packet-Switching Internetworks, Computer Communications, VOL. 15, No. 6, pages:367-373, August 1992.

[26] J. G. Gruber, Delay Related Issues in Integrated Voice and Data Networks, IEEE Transactions on Communications, VOL. COM-29, NO.6, pages:786-800, June 1981.

[27] T. Suda, H. Miyahara, and T. Hasegawa, Performance Evaluation of a Packetized Voice System Simulation Study, IEEE Transactions on Communications, VOL. COM-32, NO. 1, pages:97-102, January 1984.

[28] W. A. Montgomery, Techniques for Packet Voice Synchronization, IEEE Journal on Selected Areas in Communications, VOL. SAC-1, No. 6, pages:1022-1028, December 1983.

[29] G. Barberis and D. Pazzaglia, IEEE Transactions on Communications, VOL. COM-28, NO. 2, pages:217-227, February 1980.

[30] J. C. Bolot, H. Crepin, and A. V. Garcia, Analysis of Audio Packet Loss in the Internet, Technical Report, INRIA, 1993.

[31] I. Kouvelas and V. Hardman, Overcoming Workstation Scheduling Problems in a Real-Time Audio Tool, Technical Report, Department of Computer Science, University of College London, 1996.

[32] J. J. Bolot and A. V. Garcia, The Case for FEC-Based Error Control for Packet Audio in the Internet, Technical Report, INRIA.

[33] J. D. Detreville, A Simulation-Based Comparison of Voice Transmission on CSMA/CD Networks and on Token Buses, AT&T Bell Laboratories Technical Journal, VOL. 63, No. 1, pages:33-55, January 1984.

[34] D. Minoli, Optimal Packet Length for Packet Voice Communication, IEEE Transactions on Communications, VOL. COM-27, NO. 3, pages:607-611, March 1979.

[35] V. Paxson, End-to-End Internet Packet Dynamics, Technical Report, Network Research Group, Lawrence Berkeley National Laboratory, June 1997.

[36] T. Bially, B. Gold, and S. Senef, A Technique for Adaptive Voice Flow Control in Integrated Packet Networks, IEEE Transactions on Communications, VOL. COM-28, NO. 3, pages:325-333, March 1980.

[37] J. C. Bolot, End-to_end Packet Delay and Loss Behavior in the Internet, Technical Report, INRIA, 1993.

[38] T. Tally and K. Jeffay, A General Framework for Continuous Media Transmission Control, 21st IEEE Conference on Local Computer Networks, Minneapolis, MN, pages:374-383, October 1996.

[39] J. C. Bolot and A. V. Garcia, Control Mechanisms for Packet Audio in the Internet, Proc. IEEE Infocom '96, San Fransisco, CA, pages:232-239, April 1996.

[40] W. E. Naylor and L. Kleinrock, Stream traffic Communication in Packet Switched Networks: Destination Buffering Considerations, IEEE Transactions on Communications, VOL. COM-30, NO.12, pages:2527-2534, December 1982.

[41] D. L. Stone and K. Jeffay, An Empirical Study of Delay Jitter Management Policies, Multimedia Systems, VOL. 2, NO. 6, pages:267-279, January 1995.

[42] K. Jeffay, D. L. Stone, and F. D. Smith, Transport and Display Mechanisms for Multimedia Conferencing Across Packet-Switched Networks, Computer Networks and ISDN Systems, VOL. 26, NO. 10, pages:1281-1304, July 1994.

[43] R. Ansari and A. R. Kaye, Compressed Voice in Integrated Services Frame Relay Networks: Voice Synchronization, Technical Report, Department of Systems and Computer Engineering, Carleton University.

[44] D. L. Stone and K. Jeffay, Queue Monitoring A Delay Jitter Management Policy, Lecture Notes in Computer Science, VOL. 846, pages:149-160, Springer-Verlag, 1994.

[45] J. Suzuki and M. Taka, Missing Packet Recovery Techniques for Low-Bit-Rate Coded Speech, IEEE Journal on Selected Areas in Communications, VOL. 7, NO. 5, pages:707-717, June 1989.

[46] I. Cidon, A. Khamisy, and M. Sidi, Analysis of Packet Loss Process in High-Speed Networks, IEEE Transactions on Information Theory, VOL. 39, NO.1, pages:98-108, January 1993.