The Chinese University of Hong Kong
Computer Science and Engineering Department

Spring Semester Report of

Final Year Project **LYU9903**

QoS Schemes in Wireless Networks

Written by **Ng Maggie** (97746133)
Supervised by Prof. Lyu Rung Tsong Michael
Marked by Prof. Ng Kam Wing
Printed on April 18, 2000

Contents

1 Introduction **1**
- 1.1 Platform of Our Project 1
- 1.2 Review of Last Semester 1
- 1.3 Progress in This Semester 4
- 1.4 Division of Labour 5

2 Wireless QoS Scheme **6**
- 2.1 What Scheme We Have Thought Of 6
- 2.2 Introduction To Our Wireless QoS Scheme 7

3 Real-time Video Application **9**
- 3.1 Implementation of the Video Application (In Last Semester) 9
- 3.2 Implementation of the Video Application (In This Semester) 10
- 3.3 MPEG-1 Video Format 11
- 3.4 Resource Reservation Protocol (RSVP) 15
  - 3.4.1 What is RSVP? 15
  - 3.4.2 How RSVP works? 16
  - 3.4.3 How RSVP Works With Video Application? 18
- 3.5 RSVP Application Programming Interface 18
  - 3.5.1 RAPI Overview 18
  - 3.5.2 RAPI Client Library Calls 20
  - 3.5.3 RAPI Objects 22
- 3.6 Running Video Application 23
- 3.7 Difficulties of Implementing Video Application 26

4 Integration of Qur Project **28**
- 4.1 Integration of the whole system 28
- 4.2 Traffic Control 29

5 Performance Evaluation **35**
- 5.1 Introduction of the Experiment 35
- 5.2 Experiment Results 36
- 5.3 Analysis of The Results 37

6 Conclusion **39**
- 6.1 Improvement 39

References **41**

Chapter 1

Introduction

In this project, we have concerned about the Quality of Services (QoS) in Wireles Networks. Other than QoS in wireless LANs, we have also implemented a video application, in order to demonstrate the effects of using QoS.

1.1 Platform of Our Project

In our project, all the things are done in Linux (kernel 2.2). We choose Linux as our development environment because it is an open source operating system, and so we can fully control the kernel and easier for our development. And the wireless devices we use are WaveLAN devices produced by Lucent Technologies. WaveLAN wireless devices are compliant with IEEE802.11 standard. The PC card we use is WaveLAN IEEE Turbo (Bronze) PC card, and the access point is WavePOINT-II. The PC card we use has data rate of 2Mbps, and range of about 115 meters in indoor.

1.2 Review of Last Semester

In last year, project team LYU9802 has implemented the QoS in wired networks. The QoS scheme is mainly based on ReSource ReserVation Protocol (RSVP) and packet scheduling (CBQ). And also, they have implemented a real-time audio application that can play MP3 songs over the wired networks.

In last semester, we have continued their works. We have ported the QoS shceme into wirless networks. And we have done some performance evaluation of the QoS scheme in both wired and wireless networks. In wireless networks, we have also compared the performance on Proxim and WaveLAN wireless deviced. The application used in the performance evaluation were the real-time audio application and FTP. The following shows the summary of the performance evaluation of last semester.

Experiments on Wired Networks

In last semester, we have done the experiments on wired network. The scenario is that we have two machines collected through the wired network, one uses as server and one uses as clients. Both FTP applications and MP3 players are run in the client machine, and both applications get the source files from the same server. The quality of service is controlled on the fly. In order to see the effect of using QoS, the limit of bandwidth was set to 200Kbits.

Here is one of the experimental results. In the experiment, the limit of bandwidth was 200Kbits. QoS was using in the first measurement and no QoS was using in the second measurement. At the start of experiment, FTP was run, after 30 sec for first measurement, (and 20 sec for second measurement), MP3 player was started to run. For both measurements, the files used for FTP and MP3 player were the same and all gotten from the same server.

Figure 1.1 and figure 1.2 show the current received bandwidth by using QoS and no QoS respectively.

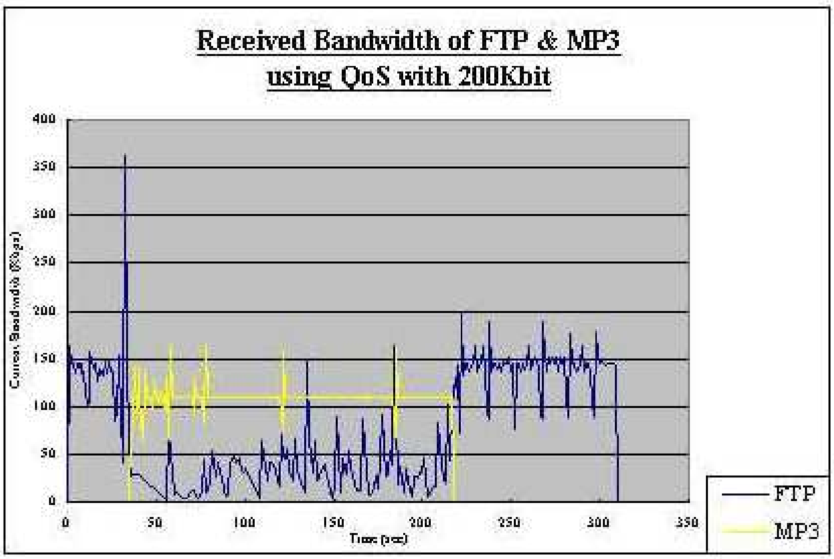

Figure 1.1: Result of the experiment: with QoS

From figure 1.1 and figure 1.2, we can see that by using QoS, MP3 played for only 3 minutes, and FTP used up only 5 minutes. But without QoS, MP3 played for about 6.5 minutes and FTP used up 6.5 minutes. And we can see from the graph that the MP3 played smoothly by using QoS, but it was not the case without QoS.

Experiments on Wireless Networks

We have also done some performance evaluation on wireless networks, by using audio and FTP applications. The scenario is similar to that on wired networks, except that in

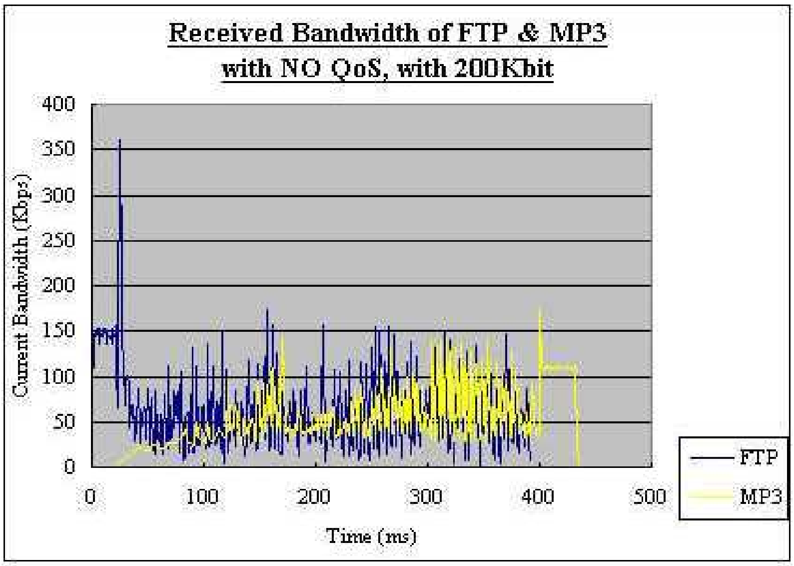

Figure 1.2: Result of the experiment: without QoS

wireless networks, the client was a wireless node.

In the experiments, the bandwidth of the network was also restricted to about 200-400Kbits. This is because, even under wireless networks, the consumption of both MP3 and FTP will not used up 2Mbits (which is the limit throughput for wireless networks), MP3 needs about 150Kbps and there is no bandwidth limit for FTP. Restriction of bandwidth is needed to investigate the performance of the system under network congestion.

Here is one of the experimental results. In this experiment, the limit of bandwidth was 200Kbits. QoS was applied in one measurement while no QoS was applied in other. At the start of experiment, FTP was run, after 30 sec, MP3 player was started to run. For all the experiments, the source of FTP and MP3 player come from the same server in wired network.

Figure 1.3 and figure 1.4 show the current received bandwidth by using QoS and without QoS respectively. The experiment was using WaveLAN wireless devices.

It is shown in the figures that by using QoS, MP3 played for about 4.46 minutes and FTP used up 7.16 minutes for WaveLAN devices. But without QoS, MP3 played for about 9 minutes, and FTP used up about 9.5 minutes. This shows that under QoS, both applications used up less time and the MP3 played more smoothly.

Analysis of The Results

In the experiments, the effect of using QoS was shown. By using QoS, the MP3 played more smoothly, and the received bandwidth of MP3 was quite steady. It was the same for both wired and wireless networks.

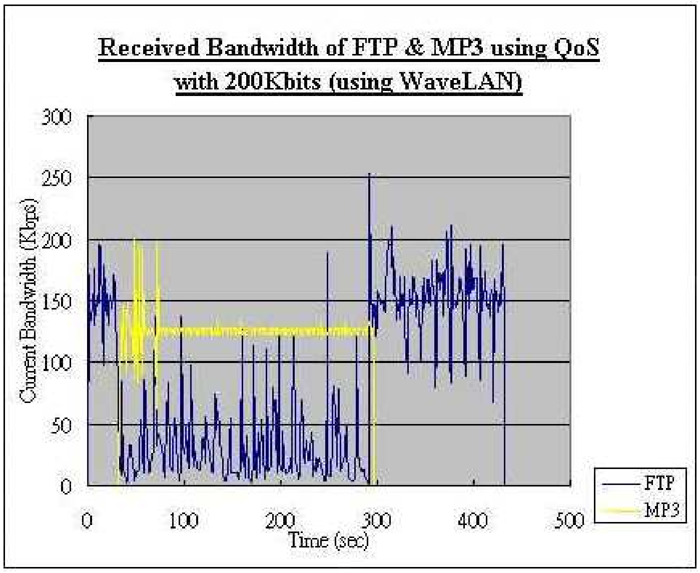

Figure 1.3: Result of experiment 1: with QoS (WaveLAN).

On the other hand, although system only ensure that MP3 (which is the real-time multimedia application) has the quality of service, in overall, both MP3 and FTP applications benefit. It is shown in the graphs that both MP3 and FTP applications run in fewer time.

And comparing the performance of applications on wired and wireless networks, we have noted that MP3 played on wired networks used up less time, no matter QoS was using or not. Thus, it was concluded that the delay on wireless network is much more than on wired network. And so the delay are a main factor to affect the performance of wireless applications.

This is the summary of the performance evaluation of using QoS in real-time audio applications, in wired and wireless networks.

1.3 Progress in This Semester

In this semester, we have implemented a QoS scheme mainly for wireless networks. Because the previous QoS scheme is mainly for wired networks, we have concerned about the disadvantages in wireless networks, and done some improvement specific for wireless networks. The mainly disadvantages of wireless networks are low reliability and long delay, variable bandwidth, and low effect of roaming. We have thought of different schemes in order to reduce these disadvantages in wireless LANs, the proposed schemes will be discussed in later chapters. And the QoS scheme we have implemented is to reduce the collision rates between the wireless nodes, and so to reduce the delay of sending packets. The scheme will also be discussed in later chapters.

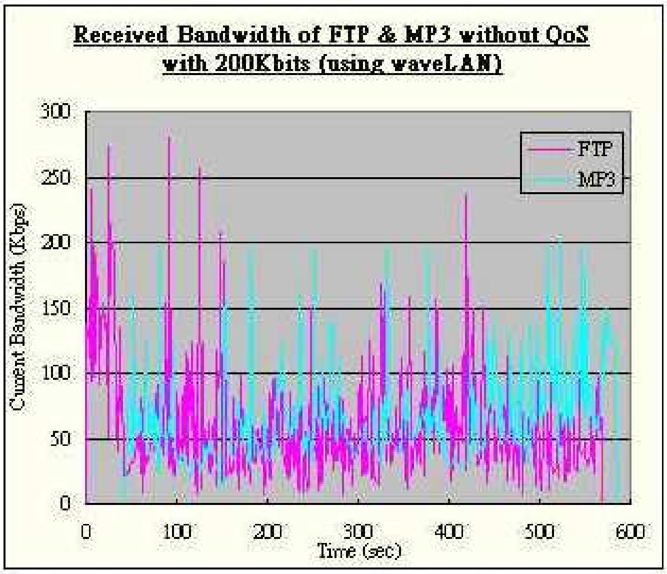

Figure 1.4: Result of experiment 1: without QoS (WaveLAN).

Besides the QoS scheme in wireless networks, we have also implemented a real-time video application on wired and wireless networks. The real-time video application can play MPEG-1 video file. It is using TCP/IP to send packets. And it is QoS enabled, so it can play the video over the networks smoothly.

1.4 Division of Labour

In the first half of this semester, I and my partner, Matchman Ho, have both thought of and discussed some QoS schemes in wireless networks. And after decided the QoS scheme used, the implementation of the QoS scheme is mainly done by Matchman.

And I am responsible for the implementation of the real-time video application. Some QoS parts in the application, that is the traffic control, is done by me and my partner.

In the following chapters, I will discuss the implementation details of our project.

Chapter 2

Wireless QoS Scheme

QoS has been talked for many years, and the shape is gradually formed. However, QoS in wireless LAN is a hot and fresh topic, many papers are come out in year 1999. Because of this, what we can take as reference is not much, as most of the papers are not very practical. So, although it may not be general enough or not very grand and great, we have thought of the schemes ourselves, and have implemented it. And we have tried our best to avoid overlapping the QoS scheme with wired LAN. What we mean is that, our scheme is suitable for wireless LAN only, and it is mainly for video/audio applications. As our method is not general, some assumptions have been taken, which will be described later. And, actually, there are many areas that can be improved for the wireless environment, but we only focus on a specific aspect.

2.1 What Scheme We Have Thought Of

I think we should tell about what we have thought of and the difficulties we meet first. In deed, we have used a few weeks to ponder what we can be done for QoS in wireless LAN. This is not an easy task, at least not as easy as thinking a product. And we must pay attention to the implementability of the scheme.

Decreasing error rate Since, according to many sources, we know that the error rate for wireless LAN is greater than that for wired LAN. So it is natural for us to think about decreasing the error rate in sending packets through the air. However, after we have searched for some papers that are talking about that, we found that what we needed to do was to alter the routine in hardware devices, which was impossible for us to do. So, we have abandoned this idea finally.

Improving the hand-off mechanism Roaming involves the communication between two access points. We have tested this on the 9th floor of Ho Sin Hang Engineering Building, and have found that the hand-off mechanism is very slow. And it is some how unacceptable. So, we have thought of improving this. Unfortunately, we have failed to find enough reference for us to do so. And access points must be involved in order to achieve this. But the problem is that, we can't alter anything about the access points, since they are hardware devices.

Using workstations to simulate access points As told above, we have thought of improving the hand-off mechanism, however access points must be involved. So, we have thought of using workstations to simulate access points. The idea is to connect an access point to a workstation. Every traffic must travel via the workstation to and from the access point. That is, using a workstation to simulate the "software" part of the hardware. The problem of this scheme is that the overhead is large, since every traffic must be filtered in the workstation.

We have thought of other schemes, but finally have been abandoned due to some reasons. In the next section, I will talk about our QoS scheme.

2.2 Introduction To Our Wireless QoS Scheme

Our QoS scheme aims at reducing the packet collision probability. The transmission medium of wireless LAN is unguided, so the packets are sent everywhere, and also, packets are come from everywhere. So, it is easy to get packets collided with other packets, at least, it is easier than that in wired LAN.

Our method is simple in speaking, that is reserving bandwidth before sending packets. One would like to reserve the bandwidth will broadcast a "signal" packet, saying that it is going to send something in the near future. When the others receive it, they will suspend sending their packets for a fixed period of time. Within this period, the one who reserved the bandwidth temporarily will send its packets. In this situation, we can ensure that the packets would not collide with others.

To implement the scheme, we have made an assumption, which is that, the environment is "clean". It means that there is no other packet or interference in the area, except the known ones. We have taken another assumption, that is, there is no wireless node entering or leaving the coverage of the access point and the two notebooks, and there should have no hidden nodes. Hidden nodes will be introduced later in the report. This method is suitable for video/audio streaming, since the packets are sent periodically, so we can predict the pattern of sending packets, and reserve the bandwidth accordingly.

The scenario of our system consists of at least two notebooks equipped with wireless devices, an access point connected to a wired network, where a server resides in. The system is in a closed area such that the environment is "clean" as to fulfill our assumption. Also, the access points and the two notebooks are within the coverages of each others.

Although the idea is simple in speaking, many prior knowledges must be known first before we can do something on this. Actually, this is the most time consuming task, that is the preparing state. The knowledges we have studied included the IEEE802.11 standard, CDMA/CA, frame strucuture of WaveLAN/IEEE, and also the networking in Linux and PCMCIA programming in Linux, etc.

The details of these topics are described in Matchman's report. To have deeper knowledge about our wireless QoS scheme, please refer to Matchman's report.

Chapter 3

Real-time Video Application

We would like to implement a real time video player that can get file through both wired and wireless networks with QoS enabled. One of the reasons of implementing real time video player is that the demand of multimedia access is increasing rapidly, it is mentioned eariler. Another reason is that real time audio needs only about 150 Kbps bandwidth, which is not enough to see the performance of multimedia on wireless LANs with 2Mbps throughputs. In order to show the effect of performance of multimedia on wireless LANs and show the significance of using QoS, implementing real time video is a good choice.

In the following sections, I will briefly describe the design and implementation of the real-time video application. The MPEG-1 video format will also be introduced. And the techniques used to enabled QoS, that is RAPI and traffic control, will be discussed. And at last, the problems faced in the implementation of video application and the performance of this application will be discussed. But first of all, I would like to describe the implementation of our video player which was done in last semester.

3.1 Implementation of the Video Application (In Last Semester)

The source of our video player is from mpeg_play, which is implemented by Berkeley Multimedia Research Center. We have hacked the source code in order to enable real time access of video files through the network. The idea of this implemetation is using RPC, we have provided some services for the client to handle the video files via network. The services are:

fopen - Provided for client to open a file.
fclose - Provided for client to close an opened file.
fread - Provided for client to retrieve data from an opened file.
fseek - Provided for client to set the position of the file pointer of an opened file.
ftell - Provided for client to get the current position of the file pointer of an opened file.
fgetc - Provided for client to get a character from an opened file.
rewind - Provided for client to reset the position of the file pointer to the start position.

By setting the environment variable LYU_MPEG_SVR to the address of the server, the modified mpeg_play will automatically retrieve the file from the server specifed. When the environment variable is not set, it will retrieve the file as normal from the local host. In last semester, the modified mpeg_play could work properly in both local and network (wired and wireless) mode. However, QoS was not currently supported. In order to enable QoS in our video application, there were still lots needed to be done.

However, we find that this version of video streaming is not good. The video cannot be played smoothly. So we have abandoned this version of video player, and implement a new one for a better result, and have made it QoS enabled. And in the following section, I am going to describe our new implementation of the video player.

3.2 Implementation of the Video Application (In This Semester)

Figure 3.1 shows the overall design of the real-time video application.

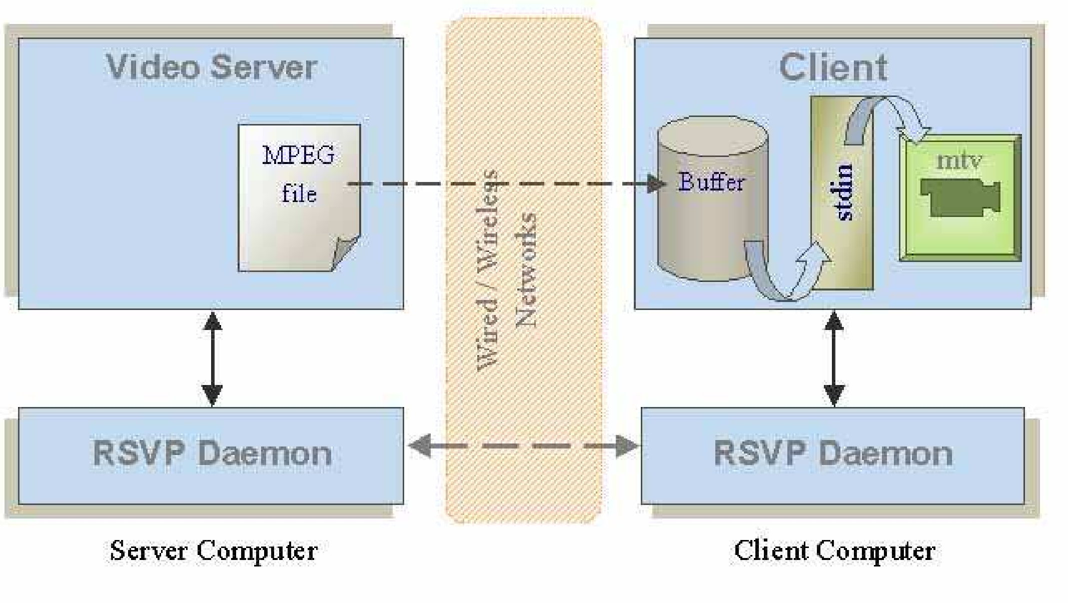

Figure 3.1: Overall Design of Real-time Video Application

In the video application, we use MPEG-1 as the video format. MPEG-1 video needs bandwidth of about 1.2Mbps. But we set the send rate of the video to be 1.3Mbps, this is because in wireless networks, the error rate and the delay is higher, we set the send rate to be a little bit higher than the required bandwidth (1.2Mbps) so that the client can received the packets in a steady rate of about 1.2 to 1.3Mbps.

The decoder used in decoding MPEG video is mtv[1]. We choose mtv as the decoder because it can play video from stdin, which is easier for the implementation of real-time player.

We have implemented a client/server model to handle the trasmission of packets through the networks. In higher view, the server will send the video file to client, which can be a wired client or wireless node, by using TCP connection. Then, the client will store the received packets into a circular buffer. On the other hand, it will put the packets into stdout, and through the pipe, the mtv player will consume the packets from stdin. By using this design, we can store the video file in server, and play the video through the networks in real-time. In our implementation, we must use TCP connection to ensure the reliable connection. It is because in wireless LANs, the error rate is high, and so it is easy to lose packets, but the decoding of MPEG file cannot has lost packets. To ensure all video packets can be transmitted to the client, TCP connection is needed.

The video application is QoS emberred. That is, by using the qos functions in server side, we can control the QoS level of the applications. The QoS used in real-time video application is RSVP and traffic control (packet scheduling). RSVP is for reservation of bandwidth. By using RAPI (API for RSVP), we can control the RSVP parameters. The traffic control is to control the priority of packets sent through the networks. Details of RAPI and traffic control will be discussed in following sections.

3.3 MPEG-1 Video Format

Before looking into the details of implementation, let's have some overview of the frame structure of MPEG-1 video[2]. By understanding the frame structure, we can have better knowledge of how to implement the video application and also know the difficulties of this implementation.

MPEG-1 video has resolution of 352x240 pixels compressed at 30 frames per second. The bandwidth needed is about 1.2Mbps. About the stream structure, in the most general form, an MPEG system stream is made up of two layers:

- System Layer

  The system layer contains timing and other information needed to demultiplex the audio and video streams and to synchronize audio and video during playback.

- Compression Layer

  The compression layer includes the audio and video streams.

In decoding, the system decoder extracts the timing information from teh MPEG system stream and sends it to the other system. And the system decoder also demultiplexes the video and audio streams from the system stream, then sends each to the appropriate decoder.

The video decoder decompresses the video stream. The audio decoder decompresses the aduio stream. The timing information extracted from system layer is used for synchronization of audio and video streams. Figure 3.2 shows the overview of general MPEG decoding system.

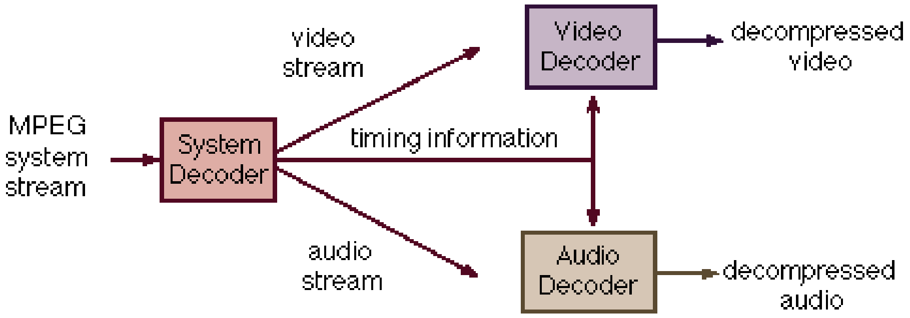

Figure 3.2: General MPEG decoding system

Video Stream Data Hierarchy

The MPEG standard defines a hierarchy of data structures in the video stream. It is shown in figure 3.3.

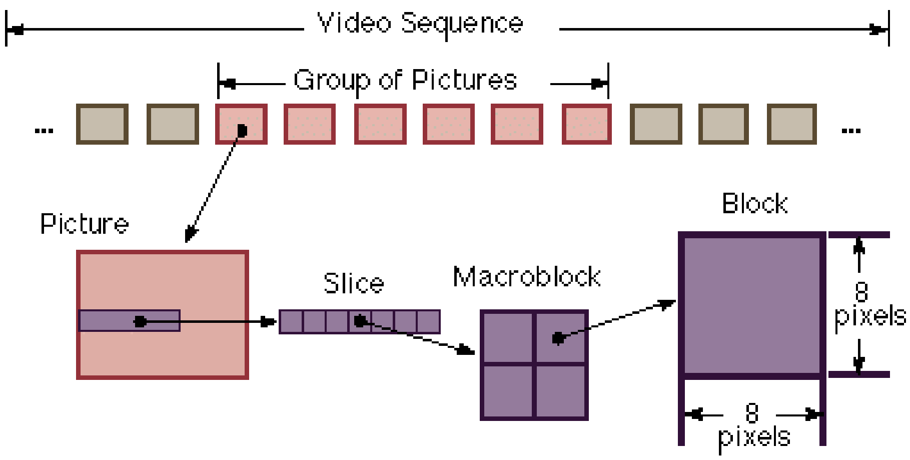

Figure 3.3: MPEG Video Stream Data Hierarchy

- Video Sequence

  It begins with one or more sequence header, includes one or more groups of pictures, and ends with an end-of-sequence code.

- Group of Pictures (GOP)

  It is a header and a series of one or more pictures intended to allow random access into the sequence.

- Picture

  The primary coding unit of a video sequence. The lower units store the information such as location of luminance and chrominance.

As much of the information in a picture within a video sequence is similar to information in a previous or subsequent picture, inter-picture coding is used. It takes the advantage of this temporal redundancy by representing some pictures in terms of their differences from other pictures. The picture types described below are the coded pictures, and so that is the actual "frame structure" in the coded MPEG video file. To understand how to transmit the MPEG video file, it is essential to undestand these coded pictures types.

The MPEG standard specifically defines three types of pictures: intra, predicted, and bidirectional.

Intra Pictures Intra pictures, or I-pictures, are coded using only infomration present in the picture itself. I-pictures provide potential random access points into the compressed video data. I-pictures use only transform coding and provide moderate compression. I-pictures typically use about two bits per coded pixel.

Predicted Pictures Predicted pictures, or P-pictures, are coded with respect to the nearest previous I- or P-picture. This is called forward prediction.

Like I-pictures, P-pictures serve as a prediction reference for B-pictures and future P-pictures. However, P-pictures use motion compensation to provide more compression than is possible with I-pictures. Unlike I-pictures, P-pictures can propagate coding errors because P-pictures are predicted from previous reference (I- or P-) pictures.

Bidirectional Pictures Bidirectional pictures, or B-pictures, are pictures that use both a pas and future picture as a reference. This is called bidirectional prediction. B-pictures provide the most compression and do not propagate errors because they are never used as reference. Bidirectional prediction also decreases the effect of noise by averaging two pictures.

The frequency and location of I-pictures can be different for different files. But I-pictures are typically used two times a second. The number of B-pictures and P-pictures can also be random. A typical arrangement of I-, P-, and B-pictures is shown in figure 3.4 in the order in which they are displayed.

However, the MPEG encoder may reorder the pictures in the video stream to present the pictures to the decoder in the most efficient sequence. Usually, the reference picutures needed to reconstruct B-pictures are sent before the associated B-pictures. Figure 3.5 shows an example of how the frames are reorder in the video stream.

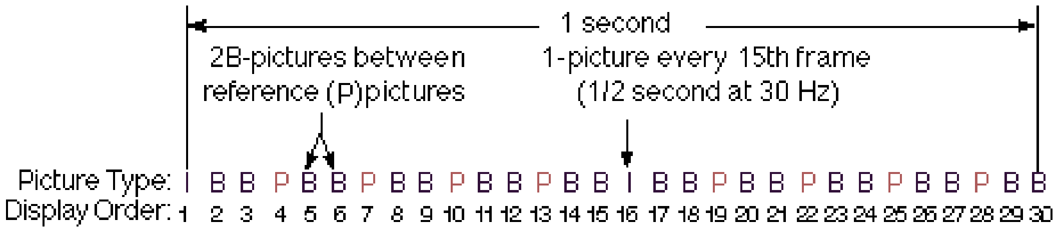

Figure 3.4: Typical Display Order of Picture Types

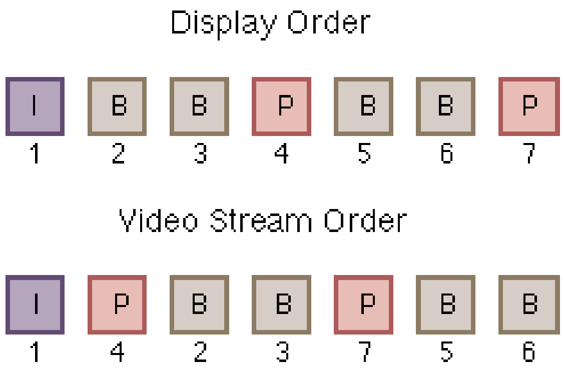

Figure 3.5: Reordering of Picture Types

Audio Stream Data Hierarchy

The MPEG standard defines a hierarchy of data structures that accept, decode and produce digital audio output. The MPEG audio stream also consists of a series of packets. Each audio packet contains an audio packet header and one or more audio frames as shown in figure 3.6. Actually, this frame structure is similar to frame structure of MP3 (MPEG-1 layer 3).

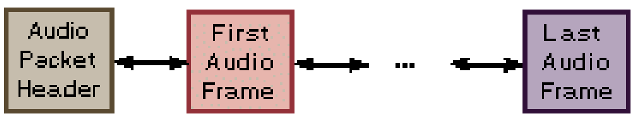

Figure 3.6: MPEG Audio Stream Data Hierarchy

Each audio packet header contains the packet start code - to identifies the packet as being an audio packet, and the packet length - to indicate the number of bytes in the audio packet.

And the audio frame contains the following information:

- Audio Frame Header contains synchronization, ID, bit rate, and the sampling frequency information.
- Error-checking Code contains error-checking information.
- Audio Data contains information used to reconstruct the sampled audio data.
- Ancillary Data contains use-defined data.

From the frame structure of MPEG video stream and audio stream, we found that it is easy to extract the information in audio stream, such as bit rate, frame size and number of frames. Because the frame structure is simple in audio stream. However, it is much more difficult and time-consuming to extract those information from video stream, as the frame structure are more complex in video stream. To extract those information, the whole file must decoded once. In order to save time during transmission of packets through the networks, we have decided not to extract the information in video stream, but we have approximately estimated the packets sending rates and size of packet in each transmission.

3.4 Resource Reservation Protocol (RSVP)

In the real-time video application, QoS is needed to ensure a good quality of video playback. RSVP is used to provide QoS. It helps to reserve bandwidth along the path from server to client. And RAPI is used in the programs to access the RSVP parameters. In the following sections, the RSVP concept is reviewed and the use of RAPI is introduced.

3.4.1 What is RSVP?

RSVP (Resource Reservation Protocol) is a protocol that allows channels or paths on the networks to be reserved for the transmission of video and other high-bandwidth messages. RSVP is part of the Internet Integrated Service (IIS) model, which ensures: best-effort service, real-time service, and controlled link-sharing. RSVP protocol is used by a host to request specific qualities of services from the network for particular application data streams or flows. It is also used by routers to deliver quality of service requests to all nodes along the path(s) of the flows and to establish and maintain state to provide the requested service.

Version 1 of RSVP has been specified in RFC2205[3] (in September, 1997) and serveral ancillary RFCs, and the Internet Engineering Task Foce (IETF) has reached consensus on the RSVP specification documents as Internet Proposed Standards.

3.4.2 How RSVP works?

To reserve the bandwidth, a host uses RSVP to request a specific QoS from the network, on behalf of an application data stream. RSVP carries the request through the network, visiting each node the network uses to carry the stream. At each node, RSVP attempts to make a resource reservation for the stream.

To make a resource reservation at a node, the RSVP daemon communicates with two local decision modules, admission control and policy control. Admission control determines whether the node has sufficient available resources to supply the requested QoS. Policy control determines whether the user has administrative permission to make the reservation. If either check fails, the RSVP program returns an error notification to the application process that originated the request. If both checks succeed, the RSVP daemon sets parameters in a packet claasifier and packet scheduler to obtain the desired QoS. The packet classfier determines the QoS class for each packet and the scheduler orders packet transmission to achieve the promised QoS for each stream. The scenerio is shown in figure 3.7.

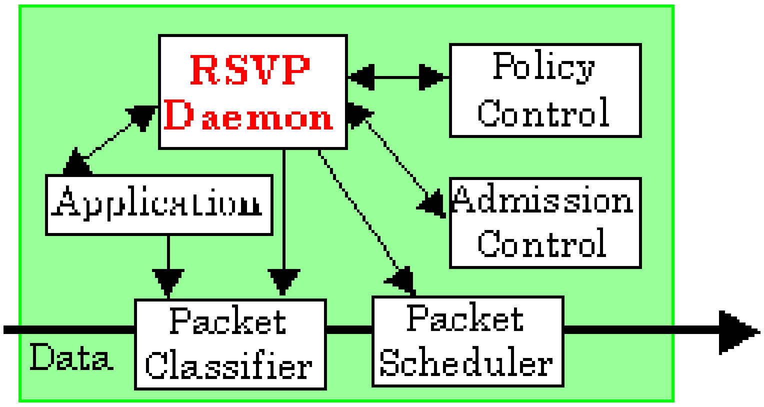

Figure 3.7: Communication of RSVP Daemon

Here is the flow of the protocol: (figure 3.8)

1. Senders characterize outgoing traffic in terms of the upper and lower bounds of bandwidth, delay, and jitter. RSVP sends PATH messages from the sender that contains this traffic specification (TSpec) information to the (unicast or multicast receiver(s)) destination address. Each RSVP-enabled router along the downstream route establishes a "path-state" that includes the previous source address of the PATH message(i.e. the next hop "upstream" towards the sender)

2. To make a resource reservation, receivers send a RESV (reservation request) message "upstream" to the source of the PATH message. In addition to the TSpec, the RESV message includes the QoS level required in an RSpec, and characterizes the packets for which the reservation is being made (e.g. the transport protocol and port number), called the "filter spec". Together, the RSpec and filter-spec represent "flow-descriptor" that routers use to identify reservations.

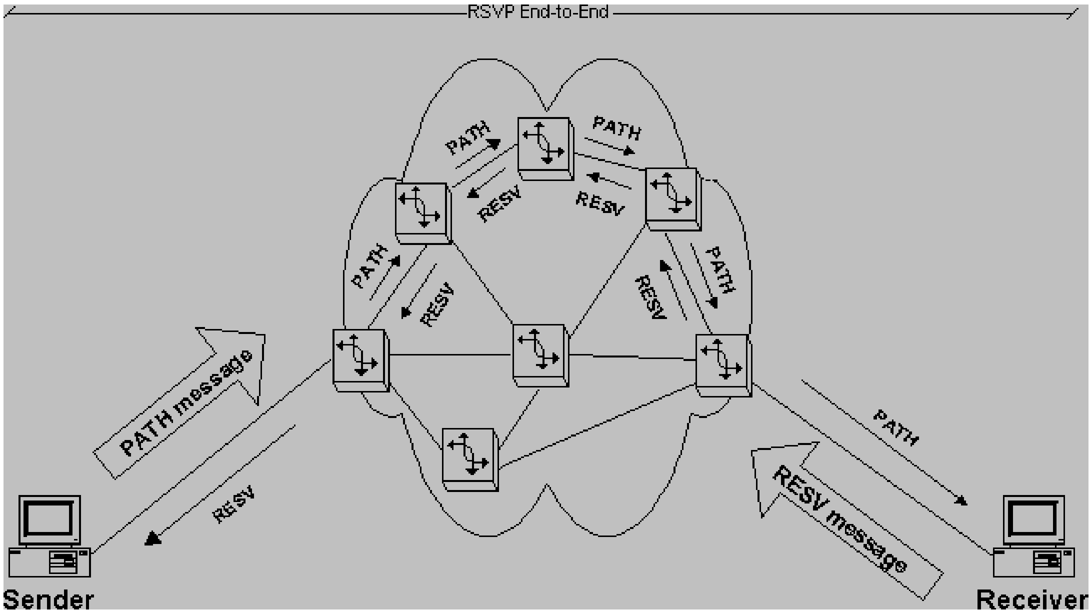

Figure 3.8: Flow of RSVP

3. When an RSVP router receivers and RESV message, it uses the admission control process to authenticate the request and allocate the necessary resources. If the request accepted, the router sends the RESV upstream to the next router.

4. When the last router receives the RESV and accepts the request, it sends a confirmation message back to the receiver.

5. There is an explicit tear-down process for a reservation when sender or receiver ends an RSVP session.

There are some characteristics that RSVP support:

- Reservations in each router are "soft", which means they need to be refreshed periodically by the receivers.

- RSVP is not a transport, but a network (control) protocol. As such, it does not carry data, but works in parallel with TCP or UDP data flows.

- Applications require APIs (RAPI) to specify the flow requirements, initiate the reservation request, and receiver notification of reservation success or failure after the initial request and trhoughout a session.

- RSVP can scale to very large multicast groups because it uses receiver-oriented reservation requests that merge as they progress up the multicast tree. The reservation for a single receiver does not need to travel to the source of a multicast tree; rather it travels only until it reaches a reserved branch of the tree.

- RSVP is very flexible, it provides opaque transport of traffic control and policy control messages, and provides transparent operation through non-supporting regions. However, this characteristic can create a "weak-link"in the QoS chain where the service falls-back to "best effort" (i.e. there is no resource allocation across these links).

- RSVP runs over IP, both IPv4 and IPv6.

3.4.3 How RSVP Works With Video Application?

In the real-time video application, RSVP is used to reserve the bandwidth for the transmission of video packets. To work with RSVP, first of all, RSVP should be installed into the Linux. The package rsvpd_linux_r4_2a3.tgz can be downloaded from [4]. The package includes RAPI, which is the interface for applications.

By using the RSVP API (RAPI), we can access the RSVP parameters in the video application. (The details of RAPI will be described later). After coding the programs with RAPI, and setting the parameters in traffic control, which will described in later chapter, QoS is enabled. By running RSVP in both client and server, as well as setting QoS enabled or disabled, we can play the real-time video through the networks with QoS. The running procedure is described in later section.

3.5 RSVP Application Programming Interface

To use RSVP, RAPI is used to request enhanced QoS. A locol RSVP control program will then use the RSVP protocol to propagate the QoS request through the routers along path(s) for the data flow. Each router may accept or deny the request, depending upon its available resources. In the case of failure, the local RSVP control program will return the decision to requesting application, such as the video application, via the API.

RSVP API implementation - RAPI[5] - is based on a client library linked with the application. RSVP is implemented on a host by a user-level daemon program. The procedures of the RSVP client library module interact with the local RSVP daemon program through a Unix-domain socket. RAPI refers to the interface between the application and the RSVP client library.

3.5.1 RAPI Overview

Using the RAPI interface, an application uses the rapi_session() call to define an API session for sending a single simplex data flow and receiving such a data flow. The rapi_sender() call may then be used to register as a data sender, and the rapi_reserve() call may be used to make a QoS reservation as a data receiver. The application can call rapi_release() to close the session and delete all of its resource reservations. The relationship among the RAPI library calls is shown in figure 3.9. rapi_sender(0) and rapi_reserve(0) means null calls in the program.

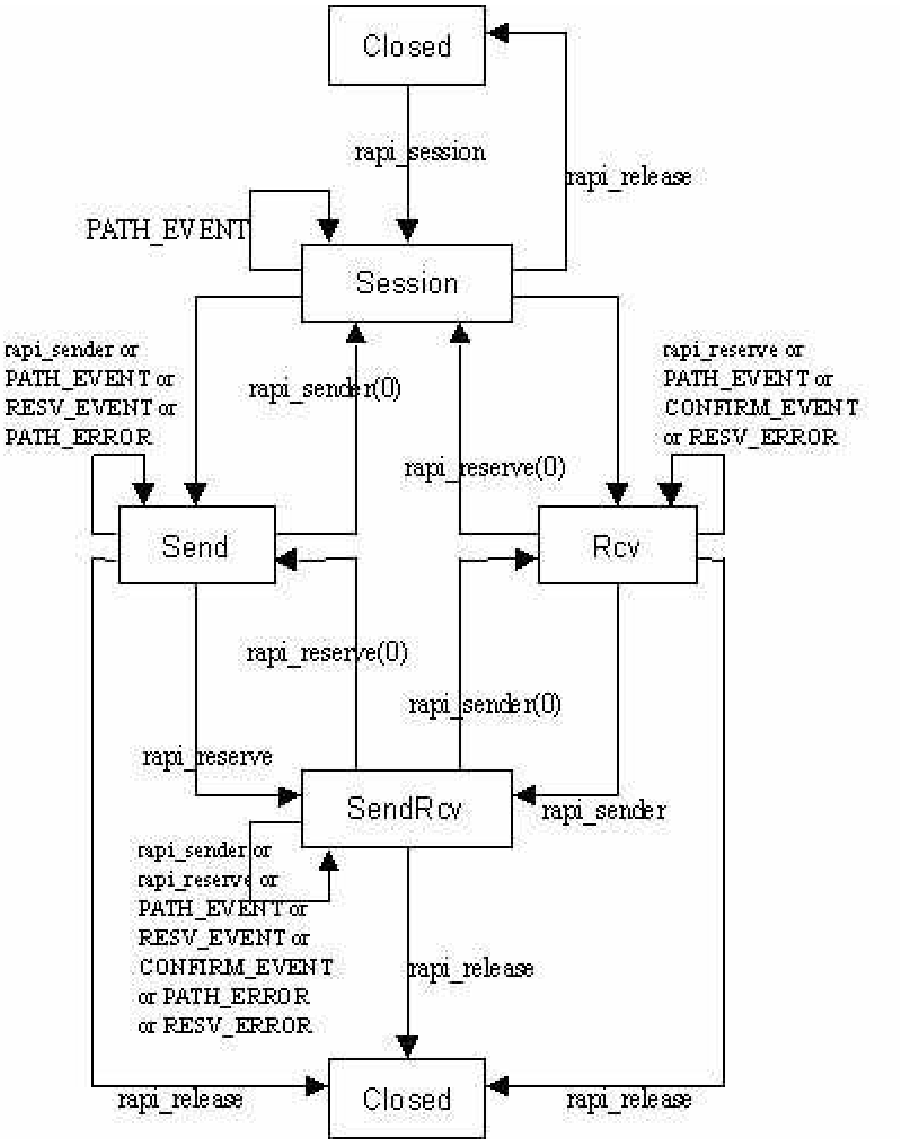

Figure 3.9: RAPI State Diagram

A single API session can define only one sender at a time. More than one API session may be established for the same RSVP session. The rapi_session() call allows the application to specify an upcall (or "callback") routine that will be invoked to signal RSVP state change and error events. There are five types of upcalls.

- PATH_EVENT and RESV_EVENT upcalls signal the arrival or change of path state and reservation state, respectively, and deliver the relevant state information to the application.
- PATH_ERROR and RESV_ERROR upcalls signal the corresponding errors.
- PATH_CONFIRM upcalls signal the arrival of a CONFIRM message.

The upcall routine is invoked indirectly by the application, using the following mechanism.

- The application issues the RAPI library call rapi_getfd() to learn the file descriptor of the Unix socket used by the API.
- The application detects read events on this file descriptor, either passing it directly in a select call or passing it to the notifier of another library, such as RPC.
- When a read event on the file descriptor is signaled, the application calls rapi_dispatch(). This drives the API to execute the upcall routine if appropriate.

The first rapi_session() call in a particular instance of the RAPI library opens a Unix-domain RAPI socket to the RSVP daemon and passes the session registration request across it. If the application (or the daemon) crashes without properly closing the RAPI socket, the other side will be notified to perform a cleanup. In particular, if the user process terminates without explicitly closing the RAPI session, the daemon will delete the corresponding reservation state from the routers.

3.5.2 RAPI Client Library Calls

The RSVPI API provides some client library calls, We can use those calls in the application by including the file *rapi_lib.h* and *rsvp_intserv.h.*

Create a Session: *rapi_session()*

The rapi_session call creates an API session. If it succeeds, the call returns an opaque but non-zero session handle for use in subsequent calls related to this API session. If the call fails synchronously, it returns zero and stores a RAPI error code into an integer variable pointed to by the *errnop* parameter.

After a successful rapi_session call has been made, the application may receive upcalls of type RAPI_PATH_EVENT for the API session.

Here is the example of usage of rapi_session():

```
/* Open RSVP Session: server (sender) */
ssnid = rapi_session(
(struct sockaddr *)&cliaddr, /* Destination host */
IPPROTO_TCP, /* IP protocol ID */
0, /* Flag: RAPI_GPI_SESSION or RAPI_USE_INTSERV */
NULL, /* Upcall routine to notify RSVP errors */
NULL, /* Arguments passed to the upcall f(x) */
&retval); /* Return RSVP error */
```

Specify Sender Parameters: *rapi_sender()*

An application must issue a rapi_sender() call if it intends to send a flow of data for which receivers may make reservations. This call defines, redefines, or deletes the parameters of that flow. A rapi_sender() call may be issued more than once for the same API session; the most recent one takes precedence. SenderTspec, SenderAdspec can be set in here.

Here is the example of rapi_sender():

```
retval = rapi_sender(ssnid, /* Session ID */
0, /* No flag */
(struct sockaddr*)&servaddr, /* Sender host */
(rapi_filter_t *)NULL, /* RAPI filter spec (optional) */
&sender_tspec, /* Tspec (traffic spec) */
&sender_adspec, /* Adspec: optional */
(rapi_policy_t *)NULL, /* Sender policy data (optional) */
0); /* TTL */
```

Make, Modify, or Delete a Reservation: *rapi_reserve()*

The rapi_reserve() procedure is called to make, modify, or delete a resource reservation for a session. The call may be repeated with different parameters, allowing the application to modify or remove the reservation; the latest call will take precedence.

An admission control failure (e.g. refusal of the QoS request) will be reported asynchronously by an upcall of type RAPI_RESV_ERROR. A *No Path State* error code indicates that RSVP state from a sender has not propagated all the way to the receiver, or it may indicate that one or more of the specified senders has closed its API and that its RSVP state has been deleted from the routers.

Here is the example of rapi_reserve():

```
retval = rapi_reserve(ssnid, /* Session ID */
RAPI_REQ_CONFIRM, /* flags */
(struct sockaddr *)&servaddr, /* Receiver host: IP & port */
RAPI_RSTYLE_FIXED, /* OK for Solaris */
(rapi_stylex_t *) NULL, /* Receiver style ext. (opt) */
(rapi_policy_t *) NULL, /* Receiver poly data (opt) */
1, /* no. of RAPI filter spec obj */
&filter_spec_list, /* filter spec list */
1, /* no. of flow spec (0=no reserve)*/
&flow_spec_list);
```

Remove a Session: *rapi_release()*

The rapi_release call removes the reservation, if any, and the state corresponding to a given session handle. This call will be made implicitly if the application terminates without closing its RSVP sessions. If the session handle is invalid, the call returns a corresonding RAPI error code, otherwise, it returns zero.

Get File Descriptor: *rapi_getfd()*

The rapi_getf() call may be used by the application to obtain the file descriptor associated with the Unix socket connected to the RSVP daemon, after a rapi_session() call has completed successfully and before rapi_release() is called. When a socket read event is signaled on this file descriptor, the application should call rapi_dispatch().

Dispatch API Event: *rapi_dispatch()*

The application should call this routine whenever a read event is signaled on the file descriptor returned by rapi_getfd(). rapi_dispatch() may be called at any time, but it will generally have no effect unless there is a pending event associated with the Unix pipe. Calling this routine may result in one or more upcalls to the aplication from any of the open API sessions known to this instance of the library.

RAPI Version: *rapi_version()*

This call returns a single integer that defines the version of the interface. The returned value is composed of a major number and an minor number, encoded as 100*major + minor. This call may be used by an application to adapt to different versions.

Upcalls

Upcalls executes when there are any PATH or RESV events occur in the RSVP.

3.5.3 RAPI Objects

Flowspecs, filter specs, sender templates, and sender Tspecs are encoded as variable-length RAPI objects. They are needed in the RAPI client library calls.

Every RAPI object begins with a header consisting of two words, the total length of the object in bytes and the type, respectively. An empty object consists only of a header, with type zero and length 8 bytes.

- Flowspecs

It stores the Integrated Services flowspec, which specify the amount of resources needed to be reserved. There are two formats for RAPI flowspecs:

RAPI_FLOWSTYPE_Simplifed

This is a "simplified" format. It consists of a simple list of parameters needed for either Guaranteed or Controlled Load service, using ther service type QOS_GUARANTEED or QOS_CNTR_LOAD, respectively. The RAPI client library routines will map this format to/from an appropriate Integrated Services data structure.

RAPI_FLOWSTYPE_Intserv

This flowspec must be a fully formatted Integrated Services flowspec data structure.

- Sender Tspecs

  It stores the Integrated Services Tspec, which specify the volume of data the sender will send. There are two formates for RAPI Sender Tspecs:

  RAPI_TSPECTYPE_Simplified and RAPI_TSPECTYPE_Intserv.

- Adspecs

  It stores the Integrated Services Tspec, which specify the volume of data the sender will send. There are two formates for RAPI Adspecs:

  RAPI_ADSTYPE_Simplified and RAPI_ADSTYPE_Intserv.

- Filter Specs and Sender Templates

  Filter Specs is used to filter out all the unwanted senders, so that only wanted information is received. There are two formats for these objects:

  RAPI_FILTERFORM_BASE and RAPI_FILTERFORM_GPI.

- Policy Data Objects

  (Not yet supported.)

Note that in an upcall, a flowspec, sender Tspec, or Adspec is by default deliverd in simplified format. However, if the RAPI_USE_INTSERV flag is set in the rapi_session() call, then the IntServ format is used in upcalls.

3.6 Running Video Application

After introduced RSVP and use of RAPI, let's talk about how to run the real-time video application.

To run the video application with QoS enabled, traffic control also needed. But this will be talked in later chapter. You can now just assume that when a command is run, the QoS can be enabled.

Server Side

Before running any application, RSVP should be set in both server side and client side. You can run the following command in root to open RSVP:

> rsvpd -R <client_address>

Rsvpd is a daemon program that uses the RSVP resource reservation protocol to set up reservation state in hosts and routers. It supports an API that allows appliations to make reservation requests. It includes an adaptation module to the admission control and traffic control mechanisms provided by the output device drivers in the kernel.

<client_address> is the IP address of the client host. It can be a hostname or a dotted-decimal number. -R parameter will make rsvpd sending a unicast copy of any UDP-encapsulated RSVP message directly to <client_address>, in addition to copies it normally sends. It ensure that the RSVP message can send to client.

Other than -R parameter, there are other parameter can be set, depends on the needs of users. Here is the usage of rsvpd:

rsvpd [-D] [-d debug_bits] [-l logging_level] [-R router_addr] [-t mcast_TTL]

-D Execute rsvpd in debugging mode.

-d Set the debugging mask to the integer debug_bits.

-l Set the logging level, which controls logging of data for debugging and diagnosis, to the integer logging_level.

-t Set the IP TTL value for multicasting diagnostic information (summary of the state) to mcast_TTL. This information is multicast when the DEBUG_MCAST_STATE bit is set in the debugging bit mask.

After running rsvpd, user can run the video server as follows:

> ./TCPsendto <port>

<port> is the port number.

To control the quality of service, user in the server side should login as root and then set the following commands:
To enable QoS:

> ./qos.wireless

To disable QoS:

> ./noqos.wireless

To delete all traffic control, you can type:

> ./del.all

To show the classfication of traffic, you can type:

> ./tc.show

qos.wireless, noqos.wireless, del.all, and tc.show are the scripts that access the parameters of traffic control. In qos.wireless and noqos.wireless, the limited bandwidth is set to 2Mbits. To limited bandwidth can be changed, in wired network, it should be set lower in order to see the effect of QoS.

In file noqos.wireless, there is no priority of any type of services. That is any applications in client requesting for the packets from the same server will compete for the bandwidth. However, in file qos.wireless, there is a priority for real-time services. We have set our real-time video application having higher priority by setting the "Type of Service" (ToS). Therefore, the video application can get more bandwidth in the transmission of packets.

The details of these scripts and traffic control is discussed in Integration Chapter.

Client Side

Same as server side, the client should first open RSVP by the following command:

> rsvpd -R <server_address>

And also, the video player (mpegTV) should also be installed in client side. You can the download the package from the official homepage[1]. The package we are using is mtv-1.0.8.14.tar.gz.

After that, if the server has already run the video server, then the client can run the video application by the following command:

> ./TCPrecv <server_address> <port>

And the client can see the mtv control box pop out as figure 3.10.

Then, just click the <File> option in the control box, and choose <Read From Stdin> is okay. See figure 3.11. The video will be play if there is enough packets in the buffer. (It is usually start when the buffer is 10% full.)

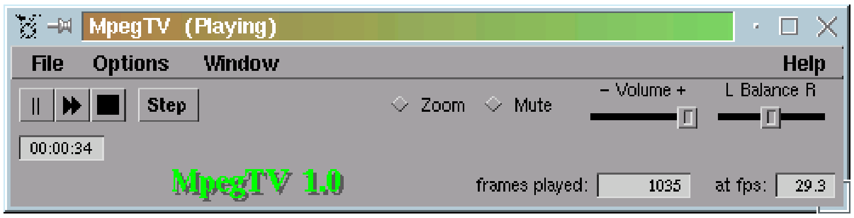

Figure 3.10: Screen Shot of Video Application (Client)

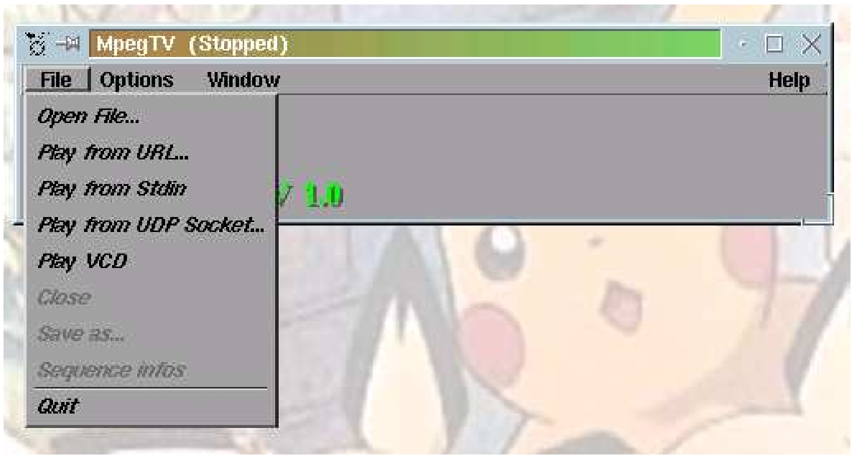

Figure 3.11: Screen Shot of Video Application (Client)

3.7 Difficulties of Implementing Video Application

Implementing the real-time video application is not as easy as imagine. In fact, in the process of implementation, we have faced some problems. First of all, as the MPEG-1 video frame structure is much more complex than MP3 audio, we cannot just scanned the MPEG video file once and get the useful information to calculate the send rate. As mentioned before, MP3 audio has simple frame structure, and so it is easy to predict the frame size, bit rates, and other information from the frame header. But it is not the case for video file. To save time in decoding the whole video file before sending to client, we have just estimated the send rate by assumming that 1.2Mbps bandwidth is needed. Although it is not as good as sending the video frame by frame, we still have adjusted the send rate so that it maintain steady send rate and can be played smoothly in client side.

Moreover, as it is hard to extract the frames from the MPEG video before decoding them, thus in the client side, we can only choose those decoders that can play video from stdin, that is mpegTV.

In the QoS part, again, we cannot extract the frame information. Thus in some parts of RAPI functions, such as filling flowspecs, we can just estimate the values. We have implemented that part by tried and error. And now the result is acceptable.

Anyway, the quality of the real-time video application is quite good. It can play the video through the Wireless networks smoothly. And with the QoS parameters plugged, it can shows the effect of QoS.

Chapter 4

Integration of Qur Project

This part is written together with my partner, as this is the common work we have done.

In this chapter, we will attempt to tell you about our idea of integrating the QoS scheme we have thought of for the wireless LAN and the QoS scheme that LYU9802 has used for the wired LAN. The traffic control in Linux will also be introduced too.

4.1 Integration of the whole system

Figure 4.1: Integration of the whole system.

Figure 4.1 shows the overall idea of our system. There are two notebooks, one being the video streaming client and the other one being the traffic source, that is, it will provide the traffic which will congest the network. The server is resided in a wired network, which will provide the video stream for the client. Access point is involved for the communication between the client, which is resided in the wireless LAN and the server, which is resided in the wired LAN. We assume that there is no other traffic across the air, and there is no wireles node leaving or entering the coverage of the access point. Moreover, the two notebooks and the access point are with the coverages of each others. LYU9802's QoS scheme will be applied for the wired part, that is from the server to the access point (actually, it is up to the client also!) while our QoS scheme will be applied for the wireless part (precisely, it is for the two notebooks).

At the very start, both the client and the server should run the RSVP daemon as introduced in the previous section in order to establish a RSVP path from the client to the server and vice versa. Then the server should start up in order to wait for the client to request video streams. After then, the client can ask the server for the video it wants. At the same time other traffic will be generated, either from the client itself or from the other notebook. Now, if no QoS scheme is applied, the client cannot see the video smoothly, and would be annoyed.

At this time, our QoS schemes are useful. First of all, for the path from the server to the client, RSVP has reserved the bandwidth for us, and the traffic control and packet schedule algorithm, that is CBQ, will give higher priority for the video stream such that the stream can reach the client punctually. And other traffic will be blocked up to some extent. However this is not enough, since the bandwidth in wireless LAN is not reliable and the collision rate is high. So, when there is no traffic for the client, it will broadcast a "signal" packet to tell the others, that is the other notebook, that it is going to receive packets soon, asking others to suspend its traffic for a while. After the other nodes receive that packet, they will be silent for a while, and the environment is cleaned up. Now, the stream from the server can reach the client safely. And the mechanism above is running over and over again, until the video is end.

The video application, RSVP, CBQ, our QoS scheme and etc. have been introduced in the previous chapters, however, one of the most important issues has not been mentioned yet, it is the traffic control in Linux, available from the iproute2+tc package. Now, we are going to introduce it.

4.2 Traffic Control

There is a package, iproute2+tc, provided for Linux to control the traffic in Linux. The installation can be refered to our previous report and [6].

According to [6], there are some problems comes along with the iproute2+tc package that the writer of it does not know. The package does not work or gives wrong result in some situation if the bugs are not fixed. In order to fix the bug, first of all, one should edit the file q_prio.c under the directory iproute2/tc/ and change the codes

```
if (pmap_mode) {
```

```
  for (idx=0; idx < TC_PRIO_MAX; idx++)
   opt.priomap[idx] = opt.priomap[TC_PRIO_BESTEFFORT];
 }
```

to

```
 if (pmap_mode) {
  for (; idx <=TC_PRIO_MAX; idx++)
   opt.priomap[idx] = opt.priomap[TC_PRIO_BESTEFFORT];
 } else
  for (idx=0; idx <=TC_PRIO_MAX; idx++)
   opt.priomap[idx] = opt.priomap[TC_PRIO_BESTEFFORT];
```

After changing the codes, one should recompile the package following the procedure given by the package itself. It order to ensure it has compiled, one should first of all, clean up the object files such that the compiler will compile the package from scratch.

Setup of a CBQ queue can be done by tc, which comes with the iproute2+tc package. However, the package is lack of documentatoin, the usage and the effect of the scripts are not clear. We have tested it with trial and error, and have read the references written by [6]. Now, we can have a little idea of what it is doing about, and have written two scripts for our QoS system. One of the scripts is to restrict the bandwidth to about 2Mbps without any QoS parameter, and the other one is to restrict the bandwidth to about 2Mbps too but with QoS applied for the video application we have implemented. The following script is the one given by the original iproute2+tc package, which is enclosed in the directory iproute2+tc/example with name cbqinit.eth1.

1. #! /bin/sh
   (Identification of shell script)

2. TC=/usr/local/src/iproute2/iproute2/tc/tc

3. IP=/usr/local/src/iproute2/iproute2/ip/ip
   (Define path of the package)

4. $TC qdisc add root dev eth0 handle 1: cbq bandwidth 10Mbit cell 8 avpkt 1000 mpu 64
   (Setup a queue using CBQ at root)

5. $TC class add dev eth0 parent 1:0 classid :1 est 1sec 8sec cbq bandwidth 10Mbit rate 10Mbit allot 1514 cell 8 weight 1Mbit prio 8 maxburst 50 avpkt 1000
   (Setup a CBQ class at root)

6. $TC class add dev eth0 parent 1:1 classid :2 est 1sec 8sec cbq bandwidth 10Mbit rate 4Mbit allot 1514 cell 8 weight 500Kbit prio 6 maxburst 50 avpkt 1000 split 1:0 defmap ff3d
   (Setup a CBQ class under root occupying the full bandwidth but will 4Mbps rate of transfer only)

7. $TC qdisc add dev eth0 parent 1:2 sfq quantum 1514b perturb 15
   (Setup a SFQ queue for the class defined in line 6)

8. $TC class add dev eth0 parent 1:1 classid :3 est 2sec 16sec cbq bandwidth 10Mbit rate 1Mbit allot 1514 cell 8 weight 100Kbit prio 2 maxburst 100 avpkt 1000 split 1:0 defmap c0
   (Setup a CBQ class under the class defined in line 6 with rate of transfer 1Mbps and full bandwidth)

9. $TC qdisc add dev eth0 parent 1:3 sfq quantum 1514b perturb 15
   (Setup a SFQ queue for the class defined in line 8)

10. $TC class add dev eth0 parent 1:1 classid :4 est 1sec 8sec cbq bandwidth 10Mbit rate 100Kbit allot 1514 cell 8 weight 10Mbit prio 8 maxburst 10 avpkt 1000 split 1:0 defmap 2
    (Setup a CBQ class under the class defined in line 9 with rate of transfer 100Kbps and full bandwidth, this is a class defined for interactive application.)

11. $TC qdisc add dev eth0 parent 1:4 sfq quantum 1514b perturb 15
    (Setup a SFQ queue for the class defined in line 10)

12. $TC filter add dev eth0 parent 1:0 protocol ip priority 100 rsvp
    (Setup a packet classifier with RSVP as its protocol at the root)

13. $IP route add 137.189.230.44 dev eth0 proto static flowid 1:4
    (Add a route entry to route table)

14. $TC class add dev eth0 parent 1:1 classid 1:7FFE cbq rate 5Mbit bandwidth 10Mbit allot 1514b avpkt 1000 weight 50Kbit prio 8 maxburst 20 cell 8
    (Setup a CBQ class under the class defined in line 6 for real time traffic)

15. $TC class add dev eth0 parent 1:7FFE classid 1:7FFF est 4sec 32sec cbq rate 1Mbit bandwidth 10Mbit allot 1514b avpkt 1000 weight 10Kbit prio 6 maxburst 10 cell 8 split 1:7FFE defmap ffff
    (Setup a CBQ class under the class defined in line 14 to redefine the real time traffic)

The explanation of the script above is given by [6].

Now, we are going to describe our scripts, which are aimed to control the traffic in our system. In order to make it narrower such that the effect of congestion can be easier demonstrated, both of the scripts restrict the bandwidths to about 2 Mbps, it is because the bandwidth for the wired LAN (Ethernet) is 10 Mbps.

noqos.wireless

```
#! /bin/sh

TC=tc
IP=ip
DEVICE=eth0
BW="10Mbit"
LIMIT="2Mbit"
REMAIN="8Mbit"

#Ethernet
$TC qdisc \
add dev $DEVICE root handle 1: \
cbq bandwidth $BW avpkt 1000 mpu 64

#Others
$TC class \
add dev $DEVICE parent 1: classid :1 est 1msec 8msec \
cbq bandwidth $BW rate $REMAIN allot 1514 maxburst 50 avpkt 1000 \
bounded isolated prio 1 split 1:0 defmap 0xffaa

#FTP and Video Stream
$TC class \
add dev $DEVICE parent 1: classid :4 est 1msec 8msec \
cbq bandwidth $LIMIT rate $LIMIT allot 1514 maxburst 50 avpkt 1000 \
bounded isolated prio 7 split 1:0 defmap 0x0001
```

qos.wireless

```
#! /bin/sh

TC=tc
IP=ip
DEVICE=eth0
BW="10Mbit"
LIMIT="2Mbit"
REMAIN="8Mbit"
VIDEO="2Mbit"
FTP="1Mbit"
```

```
#Ethernet
$TC qdisc \
add dev $DEVICE root handle 1: \
cbq bandwidth $BW avpkt 1000 mpu 64

#Others
$TC class \
add dev $DEVICE parent 1: classid :1 est 1sec 8sec \
cbq bandwidth $BW rate $REMAIN allot 1514 maxburst 50 avpkt 1000 \
bounded isolated prio 1 split 1:0 defmap 0xffaa

#IEEE 802.11
$TC class \
add dev $DEVICE parent 1: classid :2 est 1sec 8sec \
cbq bandwidth $LIMIT rate $LIMIT allot 1514 maxburst 50 avpkt 1000 \
bounded isolated prio 4 split 1:0 defmap 0x0005

#IEEE 802.11 (to be divided, declare as parent node)
$TC qdisc \
add dev $DEVICE parent 1:2 handle 4: \
cbq bandwidth $LIMIT avpkt 1000 mpu 64

#FTP
$TC class \
add dev $DEVICE parent 4: classid :3 est 1sec 8sec \
cbq bandwidth $FTP rate $FTP weight $FTP \
allot 1514 maxburst 50 avpkt 1000 \
bounded isolated prio 8 split 4:0 defmap 0x0004

#Video Stream
$TC class \
add dev $DEVICE parent 4: classid :4 est 1sec 8sec \
cbq bandwidth $VIDEO rate $VIDEO weight $VIDEO \
allot 1514 maxburst 50 avpkt 1000 \
bounded isolated prio 1 split 4:0 defmap 0x0001
```

Figure 4.2 shows the link structure used in noqos.wireless. The bandwidth we have is 10M, we must narrow it down in order to show the congestion. So, the 10M link is divided into two half, one with 8M and the other one is 2M. Now, the 2M bandwidth is shared by the FTP application and the video player. 2M is chosen since the bandwith for the WaveLAN/IEEE is about 2M. They will compete for bandwidth with each other. The result for the competition has been plotted as graph. You can see the graph in the next chapter. The "defmap" field in the script is a mask for the TOS field of the applications. The value for FTP is 5, while that for our video application is 1. So after doing a logical AND, that is 1 AND 0x0001 and 5 AND 0x0001, the result is 0x0001 for both case, therefore they will fall into the node with class id 4, as shown figure 4.2.

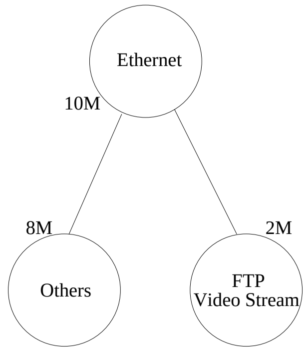

Figure 4.2: Link structure used in noqos.wireless.

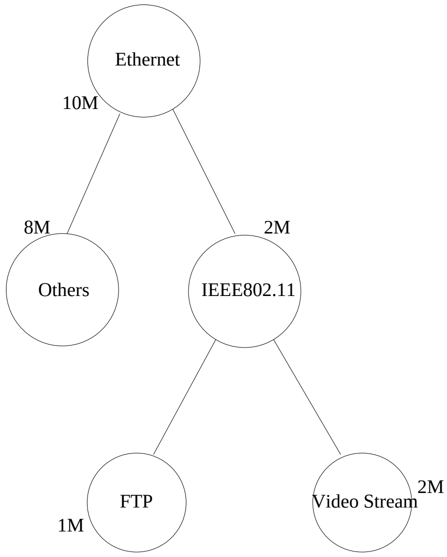

Figure 4.3: Link structure used in qos.wireless.

Figure 4.3 shows the link structure used in qos.wireless. It is based on the one in noqos.wireless. However, the IEEE 802.11 node is further subdivided so that the FTP application and video stream are treated separately. FTP application will be assigned 1M bandwidth while video stream will be assigned 2M bandwidth. You will wonder why the bandwidth is larger when summed up, that is 1M plus 2M is more than the parent node's 2M. It is because the priority is higher for the video stream to get the bandwidth (prio 1 and prio 8 in the script). So, when there is an application requesting video stream, it can get the full 2M while the FTP application will get the remaining, which is less than 1M. However, when there is no video stream sending, the FTP application can get 1M bandwidth for its transfer.

[6] has given a more detail explanation of iproute2+tc, you can refer to it if you are interested. Although they have tried out the package (which has nearly no documentation) and given some documentions, we still need to try ourselves to guess the meaning of the commands. This can be said as one of the difficult parts in our project.

Chapter 5

Performance Evaluation

5.1 Introduction of the Experiment

We have done the performance evaluation on the real-time video application. As time is limited, we have only done the performance evaluation in wireless networks. Since wireless networks has narrower bandwidth, it is much more important to have Quality of Services in wireless networks, and thus it is much more meaningful to demonstrate and evaluate the performance of real-time video application in wireless networks.

The evaluation procedure is similar to the one that was done in the last semester. The scenario of the experiment is shown in figure 5.1.

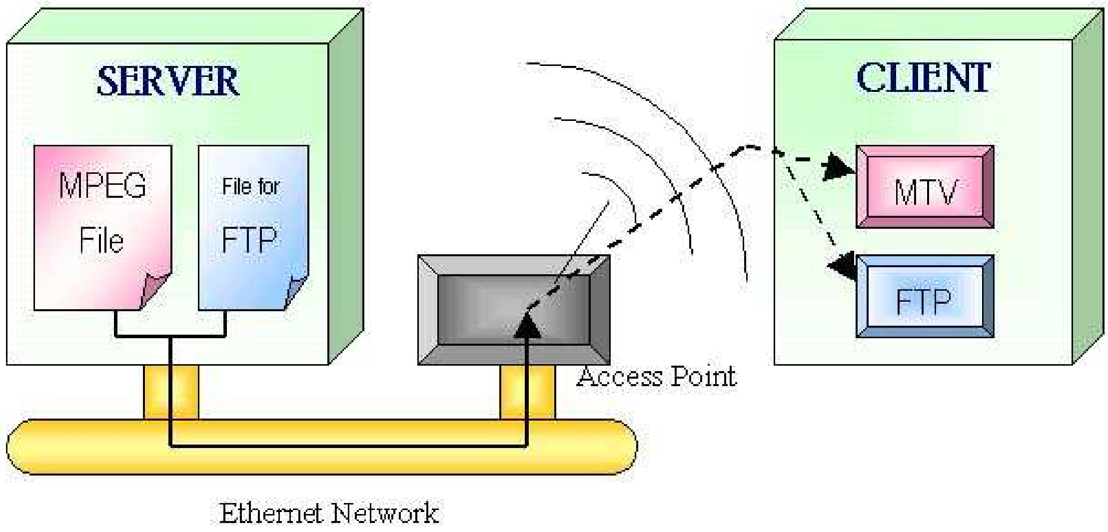

Figure 5.1: Scenerio of the Experiment

In the experiment, we have a server which is resided in the wired network. And the wired network is connected to the access point. The client, which is a wireless node, is connected to the server through the access point. We used WaveLAN wireless devices in this experiment. RSVP is used to reserve bandwidth through the path from server to client. And traffic control is done in the server side.

To evaluate the performance, we have run the real-time video application and the FTP application simultaneously. The real-time video application has higher priority than FTP to get more bandwidth. At the start of the experiment, we have run the FTP application. It requested the file transfer from a server. After some time, the real-time video application is run in the client, and it also requested the MPEG video file from the same server. During the experiment, the statistics of both FTP and video applications are record.

5.2 Experiment Results

In the experiment, we set the limit of bandwidth to 2Mbps, which is the bandwidth for WaveLAN Turbo (Bronze) PC card. QoS was disabled in the first measurement and QoS was enabled in the second measurement. At the start of experiment, FTP was run. After 30 seconds, the real-time video player was started to run. For both measurements, the files used for FTP and video player were the same and all gotten from the same server. Figure 5.2 and figure 5.3 shows the current received bandwidth without QoS and with QoS respectively.

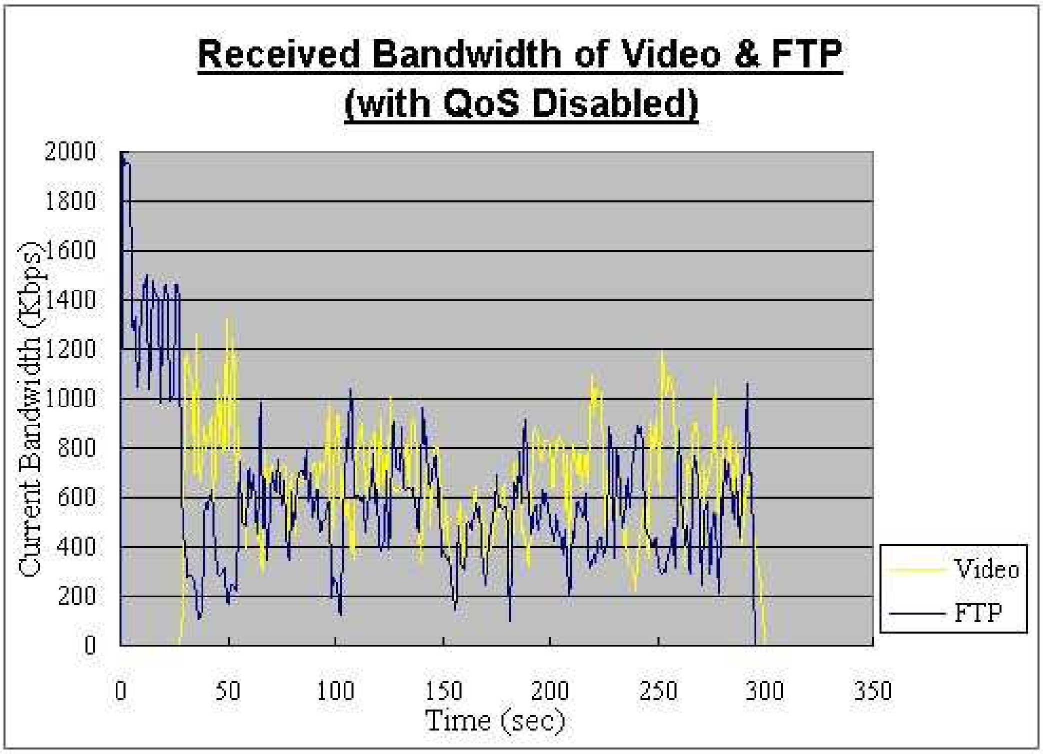

Figure 5.2: Current Received Bandwidth of Video & FTP Without QoS

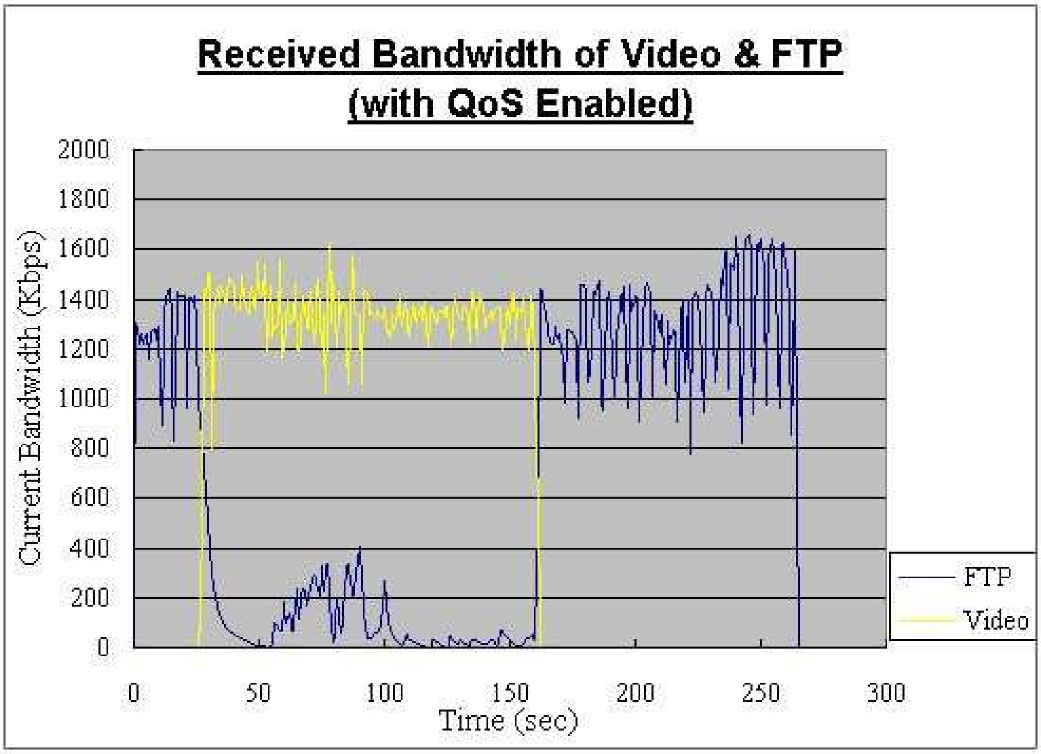

Figure 5.3: Current Received Bandwidth of Video & FTP With QoS

It is shown in the figures that by using QoS, the video played for about 2.27 minutes, and the FTP used up about 4.42 minutes. But without QoS, the video played for about 4.55 minutes, and the FTP used up about 4.91 minutes. This shows that under QoS, both applications can use up less time and the video played more smoothly than without QoS.

5.3 Analysis of The Results

In the above experiment, the effect of using QoS is shown. You can see that during using QoS, the video can play more smoothly, and it is also shown from the graphs that, the received bandwidth of video is much more steady than without QoS. This is because when QoS is enabled, real-time video application has higher priority and so bandwidth is reserved for video application. You can see from figure 5.3 that when the video is playing, the FTP application can get a little bandwidth, as most of the bandwidth is used by video application. This can ensure that the video can play smoothly in through the networks. However, when QoS is disabled, no application has higher priority and so both video and FTP applications are competing for the bandwidth. You can see from figure 5.2 that both applications have large fuctuation in received bandwidth. In the demonstration, you can see that the video application's buffer is always empty, and the video is always stopped due to not enough packets in the buffer.

Conclusion can be made from the performance evaluation that QoS is applied to wireless network successfully. The results is satisfied in both audio and video applications. And so, by using the QoS scheme, wireless network can transfer real-time multimedia data smoothly.

Chapter 6

Conclusion

In summary, what we have done in the project includes implementing a real-time video application, implementing a new QoS scheme on wireless LANs and also performance evaluation of the QoS schemes. From the performance evaluation, we concluded that the performance of QoS schemes and real-time video application are satisfiable. However, besides the schemes we have implemented, there are still many ways to improve the QoS schemes. Although the time is not allowed, we have still suggested some ways of improvement. The following shows the improvement we suggested.

6.1 Improvement

Improvement can be made in addition to what we have already done. Actually there are many other ways to provide QoS. We are going to give two improvements that can be made now.

Frame order

There are different kinds of frames in MPEG format, namely I-frame, B-frame, P-frame and D-frame. D-frame is no longer used now (and seldom if used). There usually ordering of the frames are IBBPBBPBBIBBPBBP... . I-frame is the independent frame, which does not need other frames in order to decompress itself. P-frame depends on the previous I-frame in order to decompress while B-frame depends on both the backward and foreward frames in order to decompress. So, we can improvement the streaming but sending I-frame and P-frame first and finally the B-frames. So the client can first decompress the I-frame and P-frame and at the same time receiving the B-frames. After they are decompress, the B-frames can then be decompressed without waiting for the P-frame. This is a tricky method, however, we have not yet implemented this in our system.

RAID

In server side, the video files can be stored in RAID such that the retrieval of them can be done in parallel, and therefore reducing the overhead needed to read the video streams. We can store different frames in different disks such that we can read the I-frame, P-frame and B-frames in parallel. However, synchronization should be made for the reading of frames, since I-frame and P-frame should be sent before the B-frames are sent.

Bibliography

[1] MpegTV
http://www.mpegtv.com

[2] C-Cube Microsystems - MPEG Overview
http://www.c-cube.com/technology/mpeg.html

[3] Braden, R., Zhang, L., Berson, S., Herzog, S., Jamin, S.,
Resource ReSerVation Protocol (RSVP) - Version 1 Functional Specification
RFC2205, September 1997, Proposed Standard.

[4] *http://www.cs.columbia.edu/~yhwang/cworks.htm*

[5] R. Braden, D. Hoffman *draft-ietf-rsvp-rapi-00.ps* December 15, 1997

[6] Lam Shu Yin. *Computer Science and Engineering Department (CUHK) Final Year Project (LYU9802) QoS in Linux: Traffic Control Implementation, Installation and Evaluation Final Report.*