2005

An adaptive algorithm for Internet multimedia delivery in a DiffServ environment.

Yang Feng
University of Windsor

An Adaptive Algorithm for Internet Multimedia Delivery in a DiffServ Environment

by

Yang Feng

A Thesis
Submitted to the Faculty of Graduate Studies and Research
through the School of Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Master of Science at the
University of Windsor

Windsor, Ontario, Canada
2005

Your file Votre référence
ISBN: 0-494-04940-5
Our file Notre référence
ISBN: 0-494-04940-5

Abstract

In recent years, an explosive growth of real-time multimedia applications has occurred. Many of these applications need high bandwidth. In addition, an increasing number of Internet users are using real-time multimedia applications. Thus, congestion occurs in the Internet. Also, the services required of these applications differ significantly from the traditional network applications (e-mail, file transfer, etc.) because they are delay-sensitive and loss-tolerant. Network congestion degrades the quality of real-time multimedia applications greatly.

To meet the Quality of Service (QoS) requirements of multimedia applications and to reduce the network congestion, several service models and mechanisms have been proposed. Among these, Differentiated Service (DiffServ) architecture has been considered as a scalable and flexible QoS architecture for the Internet. DiffServ provides class-based QoS guarantees. Applications in different classes receive different QoS and are priced differently. If network congestion occurs, DiffServ may not be able to guarantee the QoS for the application. Thus, the QoS may not reflect the price paid for the service. A problem of considerable economic and research importance is how to achieve a good price and quality tradeoff even at times of congestion.

This thesis presents an Adaptive Class Switching Algorithm (ACSA) which intends to provide good quality with good price for real-time multimedia applications in a DiffServ environment. The ACSA algorithm combines the techniques of Real-time Transport Protocol (RTP), DiffServ, and Adaptation together. It also takes both QoS and price into account to provide users a good QoS with a good price. The algorithm dynamically selects the most suitable class based on both the QoS feedback received and the highest user utility. The user utility is a function of quality, price, and the weight which reflects the relative sensitivity to quality and price. The class with the highest user utility is the

class that provides the best quality and price tradeoff. The QoS feedback is conveyed by RTP's Control Protocol (RTCP) Receiver Reports.

The results of simulation demonstrate that ACSA can react fast to the current class state in the network and reflects the best QoS and price tradeoff. It always seeks to find a class which provides the highest user utility except when the Internet is congested and the required QoS in all classes can not be satisfied. If this happens, the real-time multimedia flow chooses Best-Effort class with no payment.

Keyword: Differentiated Services (DiffServ), Real-time Transport Protocol (RTP), Real Time Control Protocol (RTCP), Quality of Service (QoS), User Utility, Receiver Report (RR), Adaptive Class Switching Algorithm (ASCA)

Acknowledgements

I would like express my deep gratitude and appreciation for Dr. Randa El-Marakby for her continuous support, and encouragement. She has not only been an amazing advisor for this thesis but has also been of help on academic matters. Thank You Dr. El-Marakby.

I would like to thank all my best friends, who gave me advices and helps on my thesis.

Special thanks to my husband, my parents, and my sister. Without their love, support and encouragement, this would not have been possible.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

In recent years, there has been a growing demand for real-time multimedia applications over the Internet such as telephony, videoconferencing, etc. More new multimedia products will come into the market very soon. These applications are different than the traditional text applications (e-mail, file sharing and so on). This chapter is an overview of real-time multimedia applications in the Internet. It covers the characteristics of real-time multimedia applications, the techniques for the applications, and the problems in providing QoS for the applications.

1.1 Characteristics of multimedia applications

Real-time multimedia applications have time constraints. Comparing with the traditional text applications, multimedia applications have special characteristics [Liu00].

First, real-time multimedia applications are delay-sensitive. They are valuable only if the user receives the data with a certain quality. For example, in Internet telephony, if the latency exceeds the limit that human beings can tolerate (250-400 milliseconds), users will complain about the quality of the call.

Second, many of these applications send large amount of data so they require much higher bandwidth. Meanwhile, there is an increasing number of Internet users using real-time multimedia applications. A large amount of Internet traffic load comes from multimedia applications. They are likely to bring congestion to the Internet which degrades the quality of the applications significantly.

Third, multimedia data stream is usually bursty and the receiver buffer is limited. If there is nothing done to smooth the bursty data flow*, it may overflow the receiver buffer and

* See Appendix A for the definition of flow

some data packets will be lost. Packet loss may occur due to other reasons, e.g., network congestion. Although many real-time multimedia applications are loss-tolerant, there is a threshold (about 20% loss rate) that human beings can tolerate.

1.2 Techniques for multimedia applications

Real-time multimedia applications need high bandwidth and they are delay-sensitive. These characteristics bring challenges as well as new Quality of Services (QoS) requirements to the Internet. In addition to the fast hardware development (high speed routers, high speed links, and so on), network software architectures, protocols, and algorithms for multimedia applications are very important for overcoming these challenges and satisfying the new QoS requirements. The main architectures and protocols include:

- Real-time Transport Protocol (RTP) together with its Real Time Control Protocol (RTCP) [SC03]
- Adaptive approaches.
- Real-time Streaming Protocol (RTSP) [SC98]
- Resource ReSerVation Protocol (RSVP) [BZ97, Wro97a]
- The Integrated Services (IntServ) [CSZ92, BCS94] and RSVP model
- The Differentiated Services (DiffServ) model [BB98a, BB98b]

We focus on RTP/RTCP, DiffServ, and adaptive approaches in this thesis.

1.2.1 RTP and RTCP

Most of today's real-time multimedia applications in the Internet use the Real-time Transport Protocol (RTP) for transmission of real-time multimedia data, e.g. audio and

video. RTP contains two parts: RTP Data Transfer Protocol for real-time data transmission; and RTP's Control Protocol (RTCP) for monitoring the QoS of the data delivery and providing minimal session control [SC03]. RTCP packets, especially Receiver Reports (RRs), provide feedback information, e.g., number of packets lost since the beginning of the transmission, packet loss fraction incurred in the current interval, interarrival jitter, and delay. Sender receives the feedback information from receivers and uses it to detect the current network state. Accordingly, it can dynamically adapt the transmitting rate to suit the current network state. However, this adaptation can not guarantee a minimum quality if the available bandwidth is too low.

1.2.2. Adaptive algorithms

In addition to the rate adaptation technique, there are other adaptive algorithms, e.g., dynamic bandwidth allocation [SM02, EK02, YL03], and dynamic class selection algorithm [DR01, NV00, SB02], which are used in class-based network environment. The dynamic bandwidth allocation techniques focus on maximizing the network bandwidth usage. It dynamically allocates the network resources according to the current traffic condition. The dynamic class selection algorithms focus on using minimum cost to provide acceptable QoS. It dynamically chooses the service class according to the current class state.

1.2.3 DiffServ architecture

The Differentiated Services (DiffServ) architecture has been considered as a scalable and flexible QoS architecture for the Internet. DiffServ provides class-based QoS guarantees. Packets in a DiffServ network are classified and marked differently into several classes [see Appendix A]. The forwarding behaviors of the packets in the same class are identical. Thus, DiffServ does not maintain per-flow information [see subsection 2.6] in its core. This feature allows DiffServ to achieve scalability. Basically, DiffServ provides two

classes of service: Expedited Forwarding (EF) class [JN99] and Assured Forwarding (AF) class [HB99] in addition to the Best-Effort class. EF is used by applications requiring low delay and jitter guarantees. AF is supposed to provide different levels of forwarding assurance for IP packets which require better reliability than BE service. Currently, there are four independent AF subclasses (AF1, AF2, AF3, and AF4). Each AF subclass in a DiffServ node is allocated a certain amount of resources for minimum QoS guarantees.

1.3 Problem and thesis goal

DiffServ EF and AF services provide class-based QoS and these services are associated with the cost. Usually, high-cost class provides high QoS guarantees. However, when the Internet is congested, DiffServ may not be able to guarantee the QoS for the application. Thus, the QoS may not reflect the price paid for the service. A problem of considerable commercial and research importance is how to achieve a good price and quality tradeoff even at times of congestion.

To solve the price and quality tradeoff problem, we propose an Adaptive Class Switching Algorithm (ACSA) that provides good quality with good price for real-time multimedia applications running in a DiffServ environment. The price paid for the service and the quality of the transmission are both important factors for a user to choose the service. Hence, QoS and price tradeoff should be taken into account when designing our adaptive algorithm to solve this problem.

We assume that service classes in the DiffServ architecture are ordered so that a class with a higher numerical number costs more than a class with a lower numerical number, e.g., AF4 costs more than AF3. In a normal network state, a higher price class should provide better QoS than that provided by a lower price class. But when the Internet is congested, or the traffic in some classes is overloaded, this relation may not be guaranteed. In these cases, the higher price class may not provide better service than the

lower price class. To combine the two factors (QoS and price) together, we use the user utility [SK98] function which is a function of QoS, price, and a weight that represents the user sensitivity to QoS and is determined by the user. RTCP QoS feedback report, (Receiver Reports (RRs)) [SC03], are sent by the receivers to the sender to summarize the QoS provided in the current interval for all classes. This QoS information is used to calculate the user utility for all classes. A class with the highest user utility means it is the most suitable class for the user application to run [see subsection 5.3]. By using our class switching algorithm, the user will be able to use the service class with the highest user utility.

The thesis is organized as follows. Chapter 2 provides some background about multimedia delivery. These techniques include RTP/RTCP protocol, DiffServ, IntServ architectures, and some adaptive approaches. Chapter 3 reviews the network simulator (NS) which we used to build our simulation and points out the contributed features in NS, e.g., RTP and DiffServ. Chapter 4 discusses related work. We compare the previous work to our work and summary the features of our algorithm. Chapter 5 explains our algorithm in details. It presents the problems to be solved by our algorithm, our solution, and some important terms used in our algorithm. It also analyzes the complexity of the algorithm. Chapter 6 describes our simulation topology, experimental scenarios and the results. Chapter 7 concludes the thesis and gives the future work. Appendix A provides terms used in this thesis. Appendix B is for the abbreviations.

Chapter 2 Multimedia Delivery Background

Multimedia networking plays a very important role on today's Internet. It enables different users on different machines to share image, video, and audio and to communicate with each other over the Internet. Multimedia network involves many issues and techniques. Real-time Transport Protocol together with Real-time Control Protocol (RTP/RTCP) [SC03] is suitable for transmitting and controlling applications with real-time characters. Real-time Streaming Protocol (RTSP) [SC98] provides "VCD-style" remote control for these applications. Resource ReSerVation Protocol (RSVP) [BZB97, Wro97a] is a signaling protocol to reserve resource. The Integrated Services (IntServ) and RSVP architecture, and the Differentiated Services (DiffServ) architecture are used for providing QoS for these applications. This chapter discusses these techniques in details.

2.1 Quality of Service (QoS)

Currently, the Internet's IP protocol provides Best-effort service which provides no guarantee for actual packet delivery, in order delivering and packet delay. In other words, packets could be delay, could be lost, or arrive out of order. Also, there is no differentiation between different flows within the service. The lack of guarantees and the same treatment of the flows can not meet the requirements for multimedia delivery especially for real-time multimedia applications because they need higher level of Quality of Service (QoS) guarantees than the current Internet offers.

QoS refers to the ability of a network to achieve the required functionality of an application. It can be parameterized as a set of quantitative and qualitative characteristics such as throughput, delay, jitter, packet loss and error rates, etc [Bha02, VK95]. For a particular application, different characteristics will be considered as important parameters.

The QoS parameters affecting multimedia delivery are delay, jitter, throughput, and packet loss rate [See Appendix A].

2.2 QoS for real-time multimedia applications

Transmitting the multimedia applications over the Internet needs to solve following issues [Liu00]:

- Multimedia applications means huge amount of data and heavy traffic. When bandwidth is lacking, the traffic quality will degrade due to the congestion.
- Real-time multimedia applications are very sensitive to packet delay but can tolerate some packet loss. There is no retransmitting mechanism needed for real-time applications. So the minimum bandwidth should be guaranteed when the transmission starts.
- Real-time multimedia applications need to be played back in order, and the audio and video data should be synchronized when playing back. The protocols should take into account the correct timing and synchronization.
- Multimedia applications usually work in a multicast environment. That means the same data stream from a sender is sent to a group of receivers. The protocols for multimedia applications must consider the multicast issue.

In order to solve these issues and meet the QoS requirements for real-time multimedia applications, several protocols and network architectures have been proposed including Real-time Transport Protocol together with Real-time Control Protocol (RTP/RTCP) [SC03], Real-time Streaming Protocol (RTSP) [SC98], Resource ReSerVation Protocol (RSVP) [BZB97, Wro97a], the Integrated Services (IntServ) and RSVP architecture, and the Differentiated Services (DiffServ) architecture. In this chapter, we discuss the protocols and network architectures in details.

2.3 RTP and RTCP protocols

This subsection provides an introduction to the RTP protocol and its companion protocol, RTCP.

Real-time Transport Protocol (RTP) provides end-to-end network transport functions suitable for applications transmitting real-time data, such as audio, video or simulation data, over multicast or unicast network services [SC03]. It is integrated within the application layer, typically running on top of User Datagram Protocol (UDP) [Pos80]. RTP provides services including payload type identification, sequence numbering, timestamping, and QoS monitoring. Through these services, RTP takes care of the timing and synchronization issues for real-time applications. Also, the QoS monitoring mechanism allows the sender to react to the current network state. It should be emphasized that RTP itself does not provide any mechanism to ensure timely delivery or provide other QoS guarantees. It does not guarantee delivery or prevent out-of-order delivery, nor does it assume that the underlying network is reliable and delivers packets in sequence.

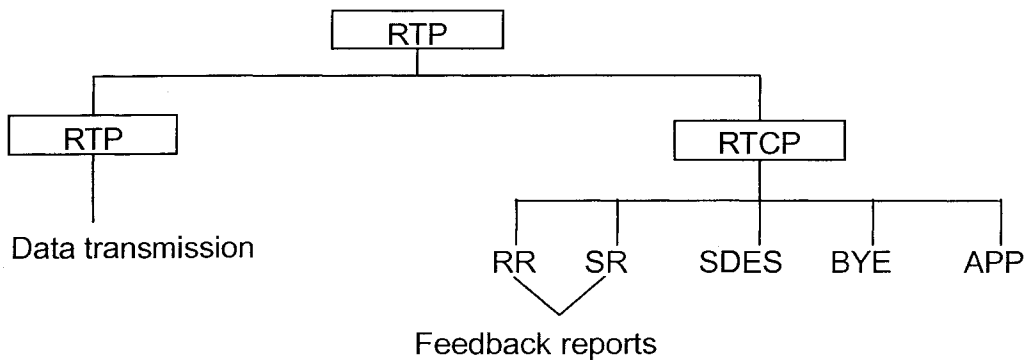

Figure 2.1 RTP protocol

RTP contains two protocols (Figure 2.1):

- RTP Data Transfer Protocol (RTP) for real-time multimedia transmission; and
- RTP Control Protocol (RTCP) for monitoring the QoS of the data delivery and

providing minimal control within a RTP session .

2.3.1 RTP Data Transfer Protocol (RTP)

RTP Data Transfer Protocol is for real-time data transmission. A RTP packet is composed of a header and the payload. The header contains several fields such as the payload type, a sequence number, a timestamp, and synchronization source identifier (SSRC), and so on.

- The payload type field (7 bits) identifies the encoding type of the RTP payload (e.g., PCM, MPEG) and determines its interpretation by the application.
- The sequence number field (16 bits) increments by one for each RTP data packet sent, and may be used by the receiver to detect the packet loss and to restore the packet sequence to be in order.
- The timestamp field (32 bits) reflects the sampling instant of the first octet in the RTP data packet. It may be used for synchronization and jitter calculations. The initial value of the timestamp is random.
- The SSRC field (32 bits) identifies the synchronization source. This identifier is chosen randomly to avoid any two synchronization sources within the same RTP session to have the same SSRC identifier. It allows the receivers to know where the data is coming from.

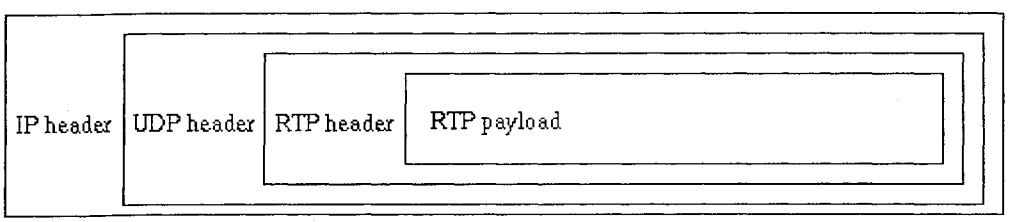

Figure 2.2 RTP data in an IP packet [Liu00]

A RTP packet includes RTP header and audio/video data (RTP payload). A RTP packet is sent to UDP socket and encapsulated to be a UDP packet. It then goes to the IP layer to be encapsulated as an IP datagram (See Figure 2.2). The encapsulation is seen only at the end systems. Routers do not know if the IP datagrams carries RTP packets or not.

RTP packets can be sent over both unicast and multicast network. In a multicast scenario, for example many-to-many multicast, each medium (audio or video) should be carried in a separate RTP session. All of the sessions typically use the same multicast group.

2.3.2 Real Time Control Protocol (RTCP)

The Real Time Control Protocol (RTCP) is based on the periodic transmission of control packets. It performs three main functions:

- QoS monitoring

  This is the primary function of RTCP. RTCP provides feedback of the quality of data transmission. Based on the feedback information, senders can estimate the current QoS of the network and adjust its transmission parameters such as sending rate or compression level. Receivers can use the feedback information as fault diagnosis to determine whether problems are local, regional or global. Also, network managers can use the feedback to evaluate the performance of their networks.

- Source identification

  RTCP SDES (source description) packets contain canonical names or CNAME as unique identifiers for RTP sources within one RTP session. They also contain other source description items such as NAME (personal name) and EMAIL (email address). Receivers require the CNAME to keep track of each participant and to associate multiple data streams from a given participant in a set of related RTP

sessions, for example to synchronize audio and video.

- Control information scaling

  RTCP packets are sent periodically among participants. When the number of participants increases, it is necessary to control the rate and limit the traffic in order for RTP to scale up to a large number of participants. In general, RTCP traffic is limited to 5 percent of the session bandwidth. To reduce the traffic, RTCP modifies the transmission rate as a function of the number of participants in the session. This number is obtained by having each participant send its control packets to all the others so that each can independently observe the number of participants.

In an RTP session, each participant sends RTCP packets to all other participants in the same session. The RTCP packets convey feedback on quality of data delivery as well as identify information about participants. In [SC03], there are five RTCP packet types that have been defined to carry a variety of control information:

- Receiver report (RR) - RRs are generated by participants that are not active data senders. They contain statistical data such as number of packets lost, interarrival jitter, and delay to indicate the reception quality of the data delivery.
- Sender report (SR) - SRs are generated by active data senders who are also receivers. In addition to the same quality feedback as in RR, SRs contain more information from senders such as number of packets sent and RTP timestamp. The timestamp is used for synchronization and jitter calculations.
- Source description (SDES) - SDES packets contain source description items to describe the sources, e.g., CNAME, NAME, EMAIL, and so on.
- Goodbye RTCP packet (BYE) – It is sent by a multicast participant to leave the session.

- Application-defined RTCP packet (APP) - Application specific functions. The APP packet is intended for experimental use as new applications and new features are developed.

In RTCP, the feedback control information is carried in two RTCP reports: Sender Reports (SRs) and Receiver Reports (RRs). SRs are generated by active senders and contain reception statistics from active senders. Receiver Reports (RRs) are generated by receivers and contain reception statistics from receivers. These statistics can be used to measure long-term and short-term QoS (delay, packet loss rate, and jitter) of the transmission. The only difference between the SR and RR forms is that the SR includes a 20-byte sender information section for use by active senders.

Some common and important fields in SR and RR are:

- The reception report count (**RC**) indicates the number of reception report blocks contained in this packet.
- The packet type (**PT**) contains a number to identify the RTCP packet type. For RTCP SR, PT is 200, for RR, PT is 201. The PTs for SDES, BYE and APP are 202, 203, and 204.
- **SSRC** is the synchronization source identifier for the originator of this report.
- **SSRC_n** is the nth SSRC identifier of the source to which the information in this reception report block pertains.
- **Fraction lost** is the fraction of RTP data packets lost from source SSRC_n since the previous SR or RR packet was sent. We refer to this field as packet loss rate or packet loss in this thesis.
- **Cumulative number of packets lost** is the total number of RTP data packets from source SSRC_n that have been lost since the beginning of reception.

- **Interarrival jitter** is an estimate of the statistical variance of the RTP data packet interarrival time.
- Delay since last SR (**DLSR**) is the delay between receiving the last SR packet from source SSRC_n and sending this reception report block.

RTCP SRs and RRs convey the reception quality feedback information which will be useful not only for data senders but also for other receivers and third-party monitors (e.g., network managers). By analyzing sender and receiver reports, the short-term or long-term network states will be learned. People may use the information to develop new algorithms or mechanisms to perform QoS control or network congestion control. The adaptive algorithm proposed in this thesis uses RRs to collect QoS of the current network state.

2.4 Real-Time Streaming Protocol

This subsection is a brief introduction to Real-Time Streaming Protocol (RTSP).

Today, multimedia data is usually sent across the network in streams. Video and audio data are broken into packets with size suitable for transmission between the servers and clients. A client can play the first packet and decompress the second while receiving the third. Thus the user can start enjoying the multimedia without waiting to the whole files. This "VCR-style" remote control for multimedia applications is performed by Real-Time Streaming Protocol (RTSP).

RTSP [SC98] is a client-server multimedia presentation protocol for providing user interactivity. It enables on-line playing and controlling the delivery of streamed multimedia data over IP network. It provides "VCR-style" remote control functionality for audio and video streams, like play, pause, rewind, and so on.

RTSP is an application-level protocol designed to work with other protocols like RTP to provide a complete streaming service over the Internet. This protocol is intended to control multiple data delivery sessions, to provide a means for choosing delivery channels such as UDP and TCP, and to delivery mechanisms based upon RTP [SC98].

RTSP messages use a different port number than the media stream. The messages start with rtsp://. Each RTSP session has a session identifier. The client starts the session with the SETUP request. The server responds with a session identifier. The session ends when the client sends a TEARDOWN request. Besides SETUP, TEARDOWN, RTSP supports other methods including PLAY, PAUSE, RECORD, OPTIONS, DESCRIBE, and so on.

RTSP is considered as "remote control" for multimedia delivering between client and server. It neither defines multimedia compression schemes, nor defines the encapsulation or the transmission of the data packets.

2.5 Resource ReSerVation Protocol

This subsection introduces RSVP protocol, its features and how it works.
RSVP [BZ97] is a signaling protocol that allows a host to request a special end-to-end quality of service for its data flows. The hosts running real-time multimedia applications use RSVP to reserve necessary resources (link bandwidth and router buffers) at routers along the transmission paths so that the requested resources can be available when the transmission actually takes place. Also, it allows hosts to establish and tear down reservation for data flows.

RSVP has three features.

- RSVP provides reservation for bandwidth in multicast trees and the reservation involves the end hosts and all intermediate routers.

- RSVP accommodates heterogeneous receivers. Different receivers on the same multicast delivery tree may have different capabilities and therefore need different QoS.
- The reservation is receiver-oriented. The RSVP reservation messages originate at the receivers and flow upstream toward the senders.

RSVP resource reservation involves several procedures [Liu00].

- **Policy control** determines whether the user has administrative permission to make the reservation.
- **Admission control** keeps track of the system resources and determines whether the node has sufficient resources to supply the requested QoS.
- The **packet classifier** determines the QoS class for each packet.
- The **packet scheduler** orders packet transmission to achieve the promised QoS for each stream.

In general, RSVP operates as follows (Figure 2.3):

- Before the application data is transmitted, the sender must set up a path by sending PATH messages to the receiver(s) specifying the QoS requirements.
- The admission routine involved in each router or host along the path decides if the new flow can be granted the requested QoS without impacting earlier guarantees for other flows.
- The receiver(s) send back RESV messages carrying reservation requests to request resources for the flow. Routers along the patch determine if the request is rejected or accepted. If the request is accepted, resources (link bandwidth and buffer space) are allocated and reserved for the flow and the corresponding flow state information is

installed in routers.

- Data is transferred from the sender to the receiver(s).

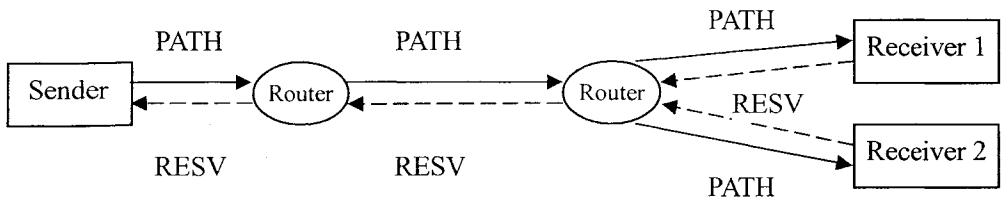

Figure 2.3: RSVP: Multicast- and receiver-oriented resource reservation

Although RSVP provides reservation for data flows, RSVP standard does not specify how the network provides the reserved bandwidth to the data flows. It does not determine which links are chosen for the reservations either. To provide the QoS guarantee, RSVP has to cooperate with other network protocols and scheduling mechanisms.

RSVP message is placed in the information field of the IP datagram. If a RSVP message is lost, a replacement refresh message should arrive soon.

2.6 Integrated Services (IntServ) and RSVP model

This subsection gives a brief introduction to IntServ [BCS94] and RSVP model including services provided in IntServ, how resources are reserved using RSVP, and drawbacks of IntServ.

Integrated Services was proposed to be the framework to provide QoS guarantees to individual flows. This model proposes two services classes in addition to the Best-Effort service: Guaranteed and Controlled-Load services. In the Guaranteed service [SP97] class, the QoS parameter values are deterministic and it guarantees both delay and bandwidth. In the Controlled-Load [Wro97b] service (also called Predictive Service) class, the QoS parameter values are estimated and it "provides the client data flow with a quality of service closely approximating the QoS that same flow would receive from an

unloaded network element", but when the network element is overloaded, it "uses admission control to assure that this service is received even" [Wro97b]. Otherwise the network provides Best-effort service class in which no QoS parameters specified.

IntServ service model needs resource reservation and it usually done by some signaling protocol (e.g. RSVP). The RSVP [BZ97, Wro97a] signaling process has been introduced in subsection 2.5. When a sender's request is accepted, the reservation information for this flow is stored in the routers along the path. When it is rejected, the network won't admit this flow.

The resource reservation and admission control in IntServ can guarantee the QoS of an application. However, they also bring problems to this service. We can see that the amount of the reservation information for each flow is proportional to the number of the flows, so in a large network, the per-flow information brings significant storage and processing overhead to routers. This architecture does not scale well in the Internet core. Also, the requirements on routers are high because all involved routers should implement RSVP, admission control, classification, and scheduling functions. Due to the scalability and manageability problem, IntServ received very limited acceptance among the network community [DR99].

2.7 Differentiated Services (DiffServ) Model

This subsection introduces DiffServ model including DiffServ overview, architecture, edge functions, core functions, and services provided in DiffServ.

2.7.1 DiffServ Overview

Today's Internet users desire service differentiation to accommodate heterogeneous application requirements and user expectations, and to permit differentiated pricing of Internet service. The DiffServ approach is introduced to provide more scalable and

manageable service differentiation for the applications than IntServ. The service model employs a small, well-defined set of building blocks from which a variety of services may be built.

In contrast to the per-flow-based QoS guarantees proposed by IntServ, DiffServ networks provide a class-based QoS assurance to achieve scalability. In a DiffServ network, packets are marked differently to create several classes, each of which can be identified in terms of DiffServ CodePoint (DSCP) located in the IP packet header [NB98]. The edge routers (either the DiffServ-capable hosts that generate traffic or the first DiffServ-capable routers that the traffic passes through) of the DiffServ domain are more complex as the packets are classified and marked at these routers. The core routers do not care about the per-flow information and simply forward packets so that DiffServ avoids the per-flow overhead and reduces the cost. That is why the DiffServ architecture is more scalable and manageable, and accepted widely in today's Internet. The forwarding is according to a Per-Hop Behavior (PHB) (See subsection 2.7.4) determined by the DSCP of the packets. The PHB provides services to allocate the buffer and bandwidth resources at each node among the competing traffic streams. Packets in the same class are treated the same way. Figure 2.4 shows a simple DiffServ domain.

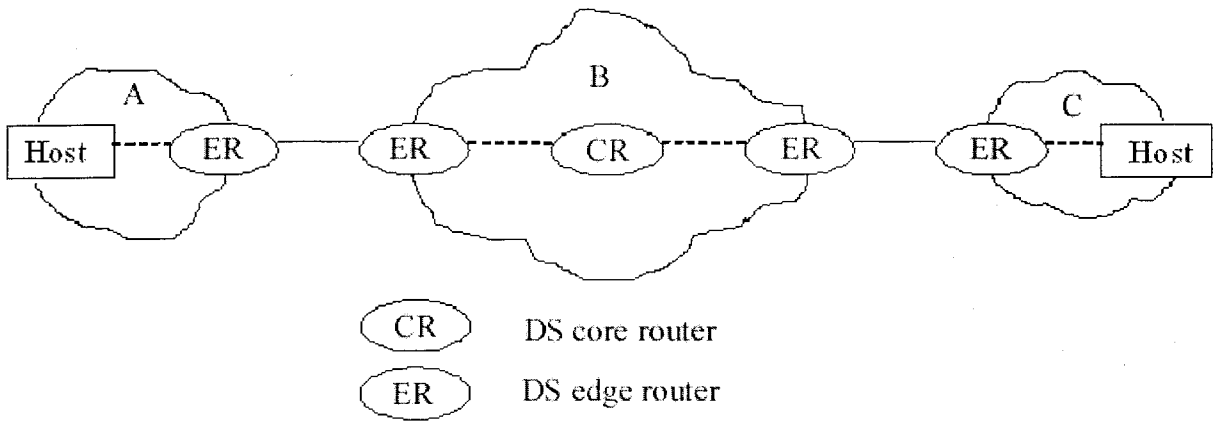

Figure 2.4 Differentiated Services Architecture

Network A and C are client domain. Network B is a backbone DiffServ domain. ERs are

DiffServ edge nodes [See Appendix A] (also called DS boundary nodes) which connect one DiffServ domain to a node either in another DiffServ domain or in a domain that is not a DiffServ domain. ERs provide DiffServ edge functions (See subsection 2.7.3). CR is DiffServ core node (also called DS interior node, DS core router) that provides DiffServ core functions (See subsection 2.7.4).

In summary, the architecture accomplishes the list of requirements as follows [Kil99]:

- Versatility: DiffServ provides a wide variety of end-to-end services which are independent of applications.
- Simplicity: The system or part of the system does not depend on the individual flow. Only a small set of forwarding behaviors is necessary.
- Cost efficiency: Information about individual flows is only used in edge routers and the states of aggregate streams are used in core routers. This reduces the cost of the network.

2.7.2 Differentiated Services CodePoint (DSCP)

In DiffServ domain, a packet is classified and marked by setting the Differentiated Services CodePoint (DSCP) in the packet header. Each DS node must use the DSCP to select the PHB to forward the packet.

The DSCP is carried in IP packet header. It is made of the six most significant bits of the IPv4 ToS (Type of Service) octet or the IPv6 Traffic Class octet [NB98]. The structure of this field is shown in Figure 2.5.

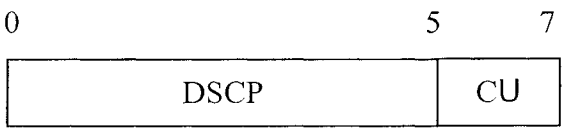

Figure 2.5 Structure of the DS field in IP header

Note: CU – currently unused subfield

2.7.3 Edge Functions – Traffic Classification and Conditioning

Differentiated services are achieved through the combination of traffic conditioning at the edge and Per-Hop Behavior (PHB)-based forwarding in the core.

Before we explain the edge functions, we need to introduce the concept of SLA – Service Level Agreement. SLA is a service contract between a customer and a service provider that specifies a forwarding service a customer should receive at a certain cost. A customer may be a user organization (source domain) or another DS domain (upstream domain) [HB99]. SLA may include traffic conditioning rules which constitute a part or whole TCA (Traffic Conditioning Agreement). TCA is an agreement specifying classifier rules, corresponding traffic profiles (a description of the temporal properties of a traffic stream such as rate and burst size), and conditioning rules (metering, marking, shaping, and policing).

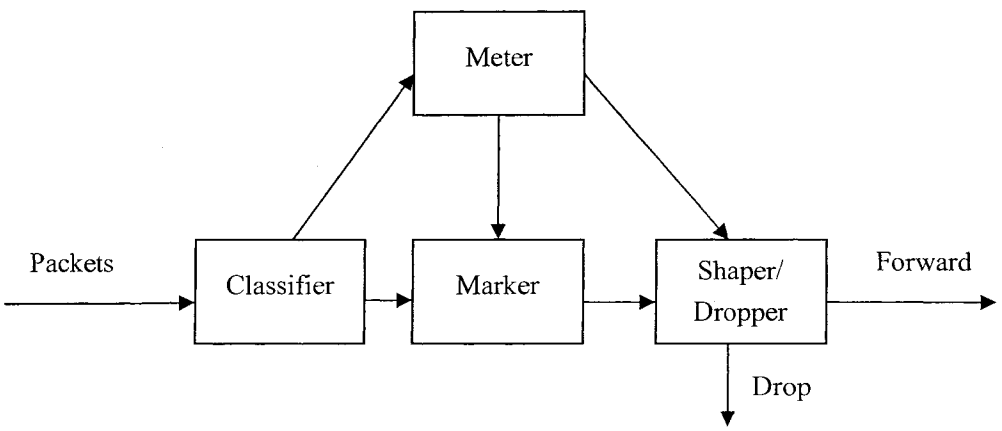

Figure 2.6 A logical view of a packet classifier and traffic conditioner

To enforce rules specified in a TCA, DiffServ architecture has traffic conditioners and classifiers. A traffic conditioner is an entity which performs traffic conditioning functions and contains meters, markers, droppers, and shapers. Traffic conditioners are typically

deployed in DS boundary nodes only. Together with a classifier which performs traffic classification function, a logical view of a packet classifier and traffic conditioner shows in Figure 2.6. Note that a traffic conditioner may not necessarily contain all four elements (meter, marker, shaper, and dropper).

The arrived packets have been classified and marked at boundary nodes by setting values of DSCP to the DS fields in the IPv4 or IPv6 packet header to determine the forwarding behavior. Then the packets will be forwarded, shaped, or dropped.

- A **classifier** selects packets based on the values of one or more packet header fields (e.g., source address, destination address, protocol ID, etc) according to defined rules and steers the packet to the meter or marker.
- A **meter** measures the temporal properties (e.g., packet sending rate, peak rate, etc) of the stream of packets selected by a classifier against a traffic profile specified in a TCA to determine whether a packet is within the negotiated traffic profile. If the packet stream is compliant with the profile, it is in-profile. Otherwise, it is out-of-profile. Out-of-profile packet might be marked differently, might be shaped, or might be dropped.
- A **marker** sets the DS field of a packet to a particular codepoint, adding the marked packet to a particular DS behavior aggregate.
- A **shaper** delays some or all of the packets in a traffic stream in order to bring the stream into compliance with a traffic profile (e.g., delays some packets so that the maximum rate constraint would be met).
- A **Dropper** discards some or all of the packets in a traffic stream in order to bring the stream into compliance with a traffic profile. This process is known as "policing" the stream.

2.7.4 Core Function - Forwarding

When the DS-marked packet arrives at a DiffServ-capable router, it will be forwarded to the next hop according to the Per-Hop Behavior (PHB) associated with that packet's class. PHB is "a description of the externally observable forwarding behavior applied at a DS-compliant node to a DS behavior aggregate" [BB98b]. Here a DS-compliant node is a node that is able to support Differentiated Services functions. DS behavior aggregate is a collection of packets with the same DSCP crossing a link in a particular direction. The PHB is determined by the DSCP and it is a service defining the forwarding behavior such as the buffer management and packet scheduling.

PHB's may be specified individually, or as a group (a single PHB is a special case of a PHB group). A PHB group usually consists of a set of two or more PHB's that can only be meaningfully specified and implemented simultaneously, due to a common constraint applying to all PHBs in the group, such as a packet scheduling or buffer management policy [NB98]. For example, in our algorithm, we use 5 PHBs in a group to provide 5 levels of services (BE, AF1 – AF4 class service). In each link, the 5 PHBs have the same packet scheduling policy (Priority scheduling).

2.7.5 Services Defined in DiffServ

Besides Best-Effort services, DiffServ provides two services: Expedited Forwarding (EF) [JN99] and Assured Forwarding (AF) [HB99]. EF is for applications requiring low queuing delay and little jitter service. AF intends to provide different levels of forwarding assurance for IP packets which require better reliability than Best-Effort Service.

Currently, the AF PHB group defines four independent AF subclasses (for simplicity, in this thesis, we will refer to the 4 subclasses as AF1, AF2, AF3, and AF4 classes). Each AF class in each DiffServ node is allocated a certain amount of forwarding resources

(buffer space and bandwidth) for minimum QoS guarantees. We refer to the minimum bandwidth as committed information rate (CIR) [see Appendix A]. The packets in one AF class must be forwarded independently from the packets belonging to another AF class. Within each AF class, an IP packet can be assigned one of three different levels of drop precedence. The drop precedence means loss probabilities. So, a packet with a lower loss probability has to be assigned to the lower level of drop precedence, and the higher loss probability has to be assigned to the two remaining levels of drop precedence. When congestion occurs, within an AF class, a router can drop the packets based on their drop precedence values.

To perform the AF services, the most widely used mechanisms are Random Early Detection (RED) [FJ93] and Multi-level RED (MRED). The RED scheme is based on detecting the congestion and notifying the congestion by dropping or marking the arriving packets. RED computes the average queue size for each output queue in order to detect congestion before the queue overflows. If the average queue size exceeds a preset threshold, it indicates congestion occurs. RED randomly chooses flows to notify of that congestion and the arriving packet is dropped or marked with a certain loss probability by setting the DSCP in the packet header.

MRED is an extension of RED. It is used to deal with the packets with multiple drop probabilities and mark the packets with different colors (green, yellow, or red). Figure 2.7 shows the MRED scheme with three drop probabilities [SM02]. In the figure, $Rmax_p$, $Ymax_p$, and $Gmax_p$ are three drop probabilities for packets marked as red, yellow, and green respectively. $Rmin_{th}$, $Ymin_{th}$, and $Gmin_{th}$ are low queue size thresholds length. $Rmax_{th}$, $Ymax_{th}$, and $Gmax_{th}$ are high queue size thresholds. The relationship of these thresholds is: $Rmin_{th} < Rmax_{th} = Ymin_{th} < Ymax_{th} = Gmin_{th} < Gmax_{th}$. Incoming packets colored with green are not dropped if the average queue length is less than $Gmin_{th}$, and these packets are dropped with probability $Gmax_p$ if the average queue length is greater

than $Gmax_{th}$. If the average queue length is between $Gmin_{th}$ and $Gmax_{th}$, green packets are dropped with a probability proportional to the average queue length. Same rule applies to packets colored with red and yellow except that the parameters are $Gmin_{th}$, $Rmax_{th}$, and $Rmax_p$ for red packets and $Ymin_{th}$, $Ymax_{th}$, and $Ymax_p$ for yellow packets.

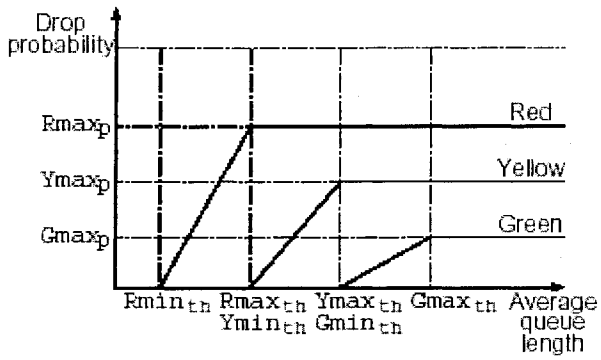

Figure 2.7 MRED scheme [SM02]

There are many policies used to provide AF service in the Internet. One is to over-provision the network so that service quality can be guaranteed under any traffic conditions. This policy needs mechanisms to handle the excessive resources allocation. The other way is to utilize the network resources more efficiently and to minimize the network costs by minimizing over-provisioning, even under-provisioning the network resources. This may cause the QoS degradation as there is a potential problem of too much traffic.

2.7.6 Problems in DiffServ

Although the DiffServ architecture is widely deployed across the Internet, it still needs to be improved to meet more and more critical requirements on networks. For example, because the DS core nodes only perform the forwarding function, there is no way for DS edge nodes to learn the internal network. That means the DiffServ is unaware of its internal network status so that it can hardly perform the desired per-class QoS when the

traffic overloads and the network state changes [LH03].

In AF services, each AF class in each DiffServ node is allocated a minimum amount of forwarding resources (committed information rate) to provide a minimum QoS guarantees. Based on the different QoS provided by DiffServ classes, applications in different classes receive different QoS and are priced differently. For example, AF4 to AF1 are four levels of services with decreasing quality, and possibly decreasing cost. In a normal network state, AF4 should always provide a service with higher QoS than or at least equal to other classes (AF3 – AF1). But when the Internet is congested, or the traffic in some classes is overloaded, this relation may not be guaranteed. In these cases, the higher price class may not provide better service than the lower price class. In an even worse case, the service provided by a higher level class such as AF4 may be worse than the one provided by a lower level class such as AF3. That means the QoS may not reflect the price paid for the service. However, the price paid for the service and the quality of the transmission are both important factors for a user to choose the service. Hence, there is a problem of how to achieve a good quality and price tradeoff and this tradeoff should be taken into account when designing the adaptive algorithm.

2.8 Adaptation

This subsection explains the concept of adaptation and several adaptation algorithms.

The Internet is a heterogeneous environment connecting various networking technologies such as Ethernet, ATM, and wireless links and so on. When delivering a multimedia application over the Internet, the available network resources will vary because of the different conditions of the networks. An efficient way to perform a desired QoS for a multimedia application is using adaptation. That means the multimedia applications are capable of adapting to changing network conditions.

For instance, the typical usage of adaptation is that the sender adapts its transmission rate dynamically based on the feedback from the receiver which reflects the current network state. Figure 2.8 shows the rate-adaptation.

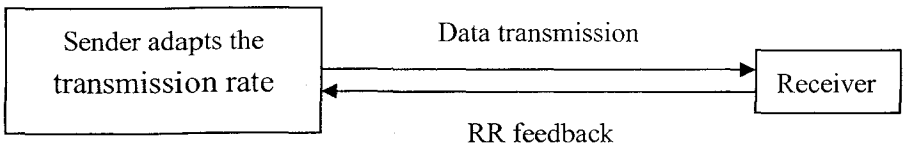

Figure 2.8 Rate-adaptation

The rate-adaptation can not guarantee a minimum quality if the available bandwidth is too low. In Differentiated Services (DiffServ) environment, dynamic bandwidth allocation [SM02, EK02, YL03], and dynamic class selection algorithm [DR01, NV00, SB02] are proposed. The dynamic bandwidth allocation approach focuses on maximum network bandwidth usage. In addition, it focuses on fairly sharing the excess bandwidth [SM02, YL03]. Here the excess bandwidth is the bandwidth left after the minimum bandwidth guarantees of all flows are satisfied. [SM02] has provided a scheme that allocates the excess bandwidth by the combination of CIR-proportional allocation and equal-share allocation. The CIR-proportional allocation part allocates the excess bandwidth proportional to their CIRs for flows with CIRs grater than zero. The equal-share allocation part allows flows in the class BE and flows with zero CIRs to utilize minimum share of the bandwidth. [EK02] focuses on the scalable pricing model which takes into account both the network users and the network providers. The proposed scalable service delivery policy based on the pricing model provides an improved level of bandwidth utilization.

The dynamic class selection algorithms focus on using minimum cost to provide acceptable QoS. Chapter 4 provides related work of dynamic class selection algorithms in details.

2.9 Conclusion

This chapter introduces the detailed techniques related to the real-time multimedia application delivery over the Internet. The topics focus on providing QoS for these applications. Among these techniques, RTP/RTCP protocol, DiffServ architecture, and adaptation are very important because our work is based on these techniques.

Chapter 3 Network Simulator

This chapter is a brief introduction to network simulator, the DiffServ model and RTP model implemented in NS. It also points out the contributed features for RTP and DiffServ in NS.

NS stands for the network simulator. The purpose of this simulator is for networking research. NS provides support for simulation over wired and wireless networks.

The network topologies defined in NS compose of routers, links and shared media. NS supports a rich set of protocols such as TCP and UDP, routing models such as static and dynamic routing, multicast protocols, applications such as FTP, HTTP, Telnet, Traffic generators, and queuing algorithms such as RED, DropTail, priority and fair queuing [NSHome].

3.1 NS architecture

NS is an object oriented simulator, written in C++ and OTcl to provide both efficiency and simplicity. The simulator supports a C++ compiled hierarchy and a similar OTcl interpreter hierarchy. The reasons of the need of the two languages are that:

- C++ is suitable for efficiently processing large data sets as it runs fast in run-time. So it is used for manipulating bytes, packet headers and for detailed protocols and algorithms implementation which need fast run-time speed.
- OTcl is suitable for those tasks which need to be tuned very often (e.g. topology configuration, setting parameters). In these cases, re-run time is important. OTcl runs slow but easy to change. Thus, it is suitable to be used to control objects and topology.

NS is a single-thread, event-driven simulator, only one event is in execution at any given time. If there is more than one event scheduled to execute at the same time, their execution is performed in a first-in-first-out manner. NS does not support partial execution of events or pre-emption.

3.2 DiffServ simulation in NS

The originally DiffServ module has been developed by Nortel Networks, but it is no longer supported by them. DiffServ is class-based QoS architecture which provides different QoS guarantees to flows. DiffServ divides traffic into different categories by marking each packet with a codepoint to indicate its category. The forwarding behaviors are according to the codepoint.

The DiffServ module in NS can support four classes of traffic. Within a single class, there are three dropping precedences allowing differential treatment of traffic. Each class corresponds to a physical RED (random early detection [FJ93]) queue, which contains three virtual queues (one for each drop precedence). A packet with a higher dropping precedence is dropped more frequently than packets with a lower precedence at times of congestion.

The DiffServ module in NS has three major components:

- **Policy**: "Policy is specified by network administrator about the level of service a class of traffic should receive in the network" [NSHome]. It defines the Per-Hop Behavior (PHB) parameters such as meter type, queuing model, rates, and burst sizes, and so on.

  In NS implementation, a policy is established between a source and destination node. All flows having the same source-destination pair are treated as a single traffic aggregate. Some contributed code has added new features to set up a policy based on

source-destination pair and the flow id in the IPv6 packet header. The Policy for each different traffic aggregate has an associated meter type, policer type, and initial codepoint. The meter type specifies the method for measuring the state variables (in- or out-of- profile, see subsection 2.7.3) needed by the policer. When a packet arrives at an edge router, it is examined and marked. The meter updates all state variables. Then the policer marks the packet (sets the codepoint) depending on these state variables. Then the packet is enqueued accordingly.

There are six policy models defined in NS and each has a corresponding policer and meter type:

- Time Sliding Window with 2 Color Marking (TSW2CMPolicer)
- Time Sliding Window with 3 Color Marking (TSW3CMPolicer)
- Token Bucket (tokenBucketPolicer)
- Single Rate Three Color Marker (srTCMPolicer)
- Two Rate Three Color Marker (trTCMPolicer)
- Nullpolicier.

- **Edge router**: Edge router marks packets with a codepoint according to the policy specified before it puts packets into the corresponding physical and virtual queues.
- **Core router**: Core router checks packets' codepoint in order to forward or drop them accordingly.

RED queue in DiffServ module

DiffServ RED queue uses Multi-level RED [MRED] scheme to handle the packets with multiple drop precedences and put packets into queues according to their codepoints.

A DiffServ RED queue provides basic DiffServ functionalities mentioned above (classifying, metering, marking, shaping or dropping). It has the following abilities:

- Implementing multiple physical RED queues along a single link;
- Implementing multiple (at most 3) virtual queues within a physical queue;
- Determining in which physical and virtual queue a packet is enqueued according to its codepoint;
- Determine from which physical and virtual queue a packet is dequeued according to the scheduling scheme chosen.

The DiffServ edge and core routers are derived from DiffServ RED queue. Edge routers have the abilities to classify, meter, mark, shape or drop packets. Core routers forward or drop packets.

Scheduler

A scheduler determines the manner in which packets are selected for transmission in a network. Currently, NS supports the following scheduling modes: Round Robin (RR), Weighted Round Robin (WRR), Weighted Interleaved Round Robin (WIRR) and Priority (PRI). Some contributed codes provides new scheduling modes such as Weighted Fair Queuing scheduling (WFQ), Self-Clocked Fair Queuing (SCFQ), Stochastic Fairness Queuing (SFQ), Low Latency Queuing (LLQ), and so on [NSCon].

Contributed features for DiffServ model

Other than the scheduling modes mentioned above, some contributed code added new features to DiffServ model [NSCon]:

- Marking for multiple flows along same source-destination path using flow id in the IP packet header (for IPv6 packets)
- Dynamically changing the policer parameters such as transmitting rate, burst sizes, and so on
- Default policy for all flows which has no particular policy defined

3.3 RTP simulation in NS

The RTP implementation in NS environment consists of three C++ classes:

- RTPAgent performs the functionalities of RTP data protocol such as sending RTP packets, Sender Reports (SRs);
- RTCPAgent performs the functionalities of RTP control protocol (RTCP) such as sending RTCP packets containing Receiver Reports (RRs);
- RTPSession manages the RTP session such as determining the time interval of sending RRs and processing RTP and RTCP packets.

The corresponding OTcl classes are Agent/RTP, Agent/RTCP and Session/RTP.

The RTP/RTCP implementation in NS is not complete, especially the implementation for RTCP reports. Contributed code by El-Marakby [EL01] provides additional functionalities including complete RTCP packet structure, RTCP sender report (SR), and Receiver Report (RR) in NS. The new RTP implementation enhances the RTP in NS and is used in our simulation.

3.4 Conclusion

Network Simulator provides a rich set of functionalities to support the simulation of wired and wireless, local and satellite networks. Among these functionalities, DiffServ module is well defined but RTP protocol needs to be extended. To simulate our algorithm, we need strong support of DiffServ architecture and RTP protocol. Although NS is weak on RTP part, adding our new functions can be easily accomplished by recompiling NS. Thus, NS is suitable for our simulation.

Chapter 4 Comparison of ACSA and Related Work

There exists some work on providing required even better QoS for real-time multimedia applications in the Internet over the past several years. This chapter discusses the previous work in the context of dynamic class selection and pricing. It also compares related work with our work.

4.1 Related work

In [SK98], Sairamesh et al have proposed a price dynamics of a vertically differentiated market which is based on the user utility function (1). The buyer makes the purchase decision according to the highest utility. If the highest utility is positive, the buyer purchases a unit from the seller who provides the utility. If the highest utility is zero or negative, the buyer does not purchase a unit from any seller.

$$U = (W(q - MIN\_Q) + (1 - W)(MAX\_P - p))\theta(MAX\_P - p)\theta(q-MIN\_Q) \quad (1)$$

(Where W is a weight that ranges between 0 and 1 and reflects the relative sensitivity to quality and price, MAX_P is the maximum price the buyer is willing to pay, MIN_Q is the minimum quality the buyer is willing to accept, q is the quality provided by a product, p is the price of the product, and $\theta(x)$ represents the step function: 1 for $x>0$ and 0 otherwise)

In this function, if price p is greater than MAX_P, then $\theta(MAX\_P - p) = 0$. If the quality q is less than MIN_Q, then $\theta(q-MIN\_Q) = 0$. At both cases, the utility is zero and the user will not buy any product. The original user utility function in [SK98] is good for analytical purpose but not for practical usage because it does not consider the balance between quality and price. The values are not normalized. We modified function (1) to function (2) by dividing the price with the maximum price (MAX_P) to make this part as a fraction and dividing the quality with the packet loss threshold (THQ) to make the two

parts have the same value range (see details in subsection 5.4). The modified utility function normalizes the values and is more suitable for the real network.

$$U = W (q_i - MIN\_Q) / THD + (1 - W) (MAX\_P - p_i) / MAX\_P \quad (2)$$

Che et al [CZ00] have proposed an analytic model which studies the impact of quality and price on the user selection of a suitable class and link bandwidth allocation. The user selection decision is calculated by a user utility function proposed in [SK98]. Therefore, the user's sensitivity to quality or to price impacts the user application switching decision. The model assumes that the quality can be measured in the same unit as the price, so it uses the original utility function. This model is an analytic model for a single link case. It has to be changed to be used in the real network where the quality cannot be measured in the same unit as the price. The authors give several numerical samples to show the impact on a single link case. With this model, a user will choose the class with the highest utility. If the highest utility is zero, he will not use any of the classes and his application will be dropped. In our algorithm, our function is justified to be used in the DiffServ environment. Also, when the highest utility is zero, we allow the user to use the best-effort class which provides services with no payment. This is more realistic than this model.

Nandagopal et al [NV00] have proposed an end-to-end delay adaptation mechanism for a core-stateless network that routers can use to dynamically adjust the class in order to match the delay bound of the flow. The algorithm seeks to achieve the delay requirement for soft real-time applications. The routers themselves provide the QoS feedback information and process it. In [NV00], each class is associated with a delay weight and the QoS of each class is measured by the delay. Each flow specifies a price that it is willing to pay and the price gets mapped to a maximum class for this flow. The authors assume a continuous range of class choices and these classes are ordered in terms of delay weight. So, when a router sees a violation of a delay requirement, it always chooses

the higher class which implies lower delay than the current one. On the other hand, when it sees the received delay is less than the required delay bound, it chooses the lower class if the delay received in this lower class satisfy the delay bound. By this way, each flow converges to a class which provides the required delay bound of this flow. When a flow is unable to get the required delay average, it will remain in its maximum class that associates with the price it is willing to pay. In our algorithm, our classes are ordered in terms of price instead of QoS (in the [NV00] case, it is delay). We measure the QoS using the packet loss rate incurred in the current interval and the QoS feedback (RTCP Receiver Reports) is provided by the end users. We believe, at times of overloading in some classes, a higher class (more expensive) may not provide better service than a lower class (cheaper). At the time of network congestion affecting all classes, our application will use the BE class but not the maximum class. We also take both price and quality into account. Therefore, the class switching will not have the same pattern as this one.

Dovrolis et al have designed a dynamic class selection (DCS) algorithm [DR01] particularly used in a relative differentiation model [DR99] to provide absolute QoS guarantees. Relative differentiation model is not same as DiffServ model that our algorithm uses. Relative differentiation model gives the assurance that services provided by higher classes are better than lower classes. The algorithm computes the minimum acceptable class selection (the lowest class which satisfies the absolute QoS requirement) for each user. If the class can be found, the user will converge to this minimum acceptable class. Otherwise, the user will stay in the highest class even when the requirement is unsatisfied. Interestingly, by selecting different values of some factors in the algorithm, the algorithm can control the tradeoff between the performance and cost of a flow. As this is a complex algorithm, to control the performance and cost tradeoff, many parameters should be considered. Our algorithm uses an efficient way to control the quality and price tradeoff by adjusting only the weight of the quality in the user utility function. Moreover, in our algorithm, we assume that the higher class may not always

provide better service than the lower class when the higher class is overloaded or at times of congestion. Also, if the user is very sensitive to the price, a higher class may not be a better choice. Another different result in our algorithm is that, when the quality requirement cannot be satisfied in all classes, the user will choose BE class instead of staying in the highest class with high cost.

Scheidegger et al [SB02] have presented the class switching algorithm (CSA) which allows a real-time application always to select the lowest and cheapest service class that still can achieve the QoS requirements in a DiffServ environment. [SB02] assumes the classes are ordered and higher order classes should always provide a service that is "at least equal or better than" lower order classes. Under this assumption, when the QoS of an application in current class is satisfied, to reduce the cost, the algorithm probes the next lower service class only and the application switches to it if the QoS of the lower class is sufficient. When the QoS of the current class is violated, the algorithm chooses the next higher class directly without probing the QoS of the class. If this higher class still cannot satisfy the QoS requirement of the application, the application will continue to switch to the next higher class until this is the highest class. Therefore, if the highest class cannot satisfy the QoS requirement, the algorithm has the same result as in [DR01] that the application will stay running in the highest class unsatisfied with high cost. Thus, this algorithm is good for the well-provisioned DiffServ network with no congestion in higher classes. This algorithm does not control the price and quality tradeoff. Comparing this algorithm with our ACSA algorithm, the ACSA algorithm uses the user utility function and puts weight for the quality to control the price and quality tradeoff. We probe all classes simultaneously and the application will switch to the most suitable class directly. This accelerates the application to converge to the suitable class. Also, our algorithm works in a more general DiffServ environment. No matter if the higher class is overloaded or the network is congested, our algorithm allows the application to switch to the class with the best quality and price tradeoff.

4.2 Features of our work

Our algorithm intends to work in a more general Differentiated Services (DiffServ) environment. No matter whether the network is congested or not, it seeks the most suitable class that provides the best quality and price tradeoff and let the application switch to it.

In summary, our algorithm has the following features:

- It is a combination of RTP, DiffServ, and Adaptation

  The algorithm uses DiffServ AF and BE services to provide a means for QoS control for real-time multimedia applications.

  The algorithm uses only RTCP Receiver Reports (RRs) feedback information to detect the current network state. We use packet loss rate conveyed in RRs to reflect the current QoS received in classes because experimentation shows that the packet loss is the dominating factor to make the class switching decision [SB02].

- It is a combination of both quality and price

  In addition to the QoS (here it is measured by packet loss rate carried by RRs), the switching is also based on the price of the service, i.e. based on the tradeoff between QoS and price. The algorithm provides a good quality and price tradeoff for real-time multimedia applications

- It works in a general DiffServ environment - even at times of congestion

  The classes are ordered in terms of price. In a normal network state, a higher price class should provide a better QoS than that provided by a lower price class. But when the Internet is congested, or the traffic in some classes is overloaded, this relation may not be guaranteed. In our algorithm, we use the user utility function to

calculate the utilities for all classes and the switching is based on the highest utility. If a higher class provides worse quality then a lower class, the application may switch to the lower class. If the highest utility is zero, the application switches to BE class with no payment.

The algorithm probes the QoS in all classes simultaneously so that the application can choose the most suitable class directly and switch to it.

4.3 Conclusion

This chapter discusses the related work in the context of dynamic class switching. It compares related previous work with our work and presents a summary of our work. The main contribution of our work is that our algorithm is a combination of price and quality. It provides good quality and price tradeoff for real-time multimedia applications running in a DiffServ environment even at times of congestion.

Chapter 5 Adaptive Class Switching Algorithm

This chapter explains our algorithm in details. It presents the problems to be solved by our algorithm, our solution, and some important terms used in our algorithm. It also analyzes the complexity of the algorithm.

5.1 Overview

We consider four AF [HB99] classes (AF1, AF2, AF3, and AF4) provided in a DiffServ architecture and the Best-Effort (BE) class in our algorithm. As the QoS of data flows running in Expedited Forwarding class (EF) [JN99] has strict guarantees, this class is out of our scope.

The goal of our Adaptive Class Switching Algorithm (ACSA) is to provide customers a good service with a good price and to control the tradeoff between quality and price. This is accomplished by taking both quality and price into account. The algorithm uses the current QoS feedback in all considered classes (BE and AF classes) to calculate the user utilities [SK98] in these classes and the switching decision is based on the highest utility. The user utility is a function of quality, price, and the weight which reflects the relative sensitivity to quality and price. The quality is measured using the fraction of packet loss (i.e., the packet loss incurred in the current interval) carried by RTCP Receiver Reports (RRs). The weight is negotiated between the user and the Internet Service Provider (ISP) and assigned by the ISP. We believe that by using our algorithm, the user will be able to use the service class which provides a good quality and price tradeoff. Hence, there will be no bad service at high price.

In our algorithm, we assume that BE and AF1..AF4 classes are ordered in terms of price and denoted by class 0 to class 4. Class 0 refers to the BE class, while classes 1..4 refer to AF1..AF4 classes respectively. The following are more assumptions:

- The price is predetermined for each class.
- A class with a high numerical number has a higher price than a lower numerical number class1, e.g. class 2 has a higher price than class 1.
- The price for Best-Effort service is 0.
- The user is charged according to the class that his application is using and according to the number of bytes/s sent.

When a RR arrives, the sender stores the packet loss rate for the current interval carried in the RR into a table. The packet loss rate is used to calculate the user utility which is a function of quality, price, and the weight of quality. The sender updates the packet loss rates table each time when there is a new RR arrives. Only the RR reporting feedback of the quality of the real-time multimedia data flow generates a call to the algorithm. Then the algorithm uses the packet loss rates stored in the table to calculate the user utilities of all classes and get the class with the highest utility. If this class is not equal to the current one, the application[2] switches to this class. If the highest utility is zero, which means the network is congested, the application switches to the BE class with no cost.

Because the user utility is a function of quality, price, and the weight of the quality, the class with the highest utility provides the best quality and price tradeoff. Switching based on the highest utility provides a good service with a good price. Also, by adjusting the weight of the quality, the algorithm can control the tradeoff between quality and price.

5.2 Packet loss threshold (THQ) and QoS

The current packet loss incurred is calculated by the receiver and sent to the sender in an

[1] Through out the thesis, we refer the lower/higher numerical number class to the lower/higher class and refer the lowest/highest numerical number class to lowest/highest class.

[2] Through out the thesis, we use the word application to mean the real-time multimedia data flow that will be running in different classes.

RR feedback report. The current packet loss rate indicates the recent quality received by the receiver. If the loss rate is large, the quality is bad, and vice versa. We use the packet loss rate to indicate the QoS of the data transmission as it is the dominating factor to make the class switching decision [SB02]. We assume that a packet loss threshold (THD) is negotiated by the user and the Internet Service Provider (ISP) when signing the Service Level Agreement (SLA) [see Appendix A]. In our simulation (see chapter 6), we simulate an Internet voice application. We set THD = 0.2 as 0.2 is the threshold that a human being can tolerate in a voice application. If the application experiences packet loss greater than THD, then the quality is considered bad. We use function (1) to calculate the quality received by the receiver in class i.

$$q_i = 1 - currPktLoss_i \qquad (1)$$

Where

- q_i is the QoS received by the receiver in class i
- $currPktLoss_i$ is the current packet loss experienced by the application running in class i.

Suppose our real-time multimedia application is currently using the current class, denoted by currClass. When the sender receives a new RR from the receiver, it saves the most recent packet loss rate of the class currClass into a table. Meanwhile, the sender sends helper packets to other classes to probe the transmission quality of these classes. When the sender receives new RRs from the other classes, it also stores the packet loss rate experienced in the other classes into the table. Hence, the sender always keeps the most recent QoS information of the BE and AF classes in the Internet. The sender uses the packet loss incurred to compute user utilities for all classes.

5.3 User utility (U)

The user utility [SK98] is a function of quality (q), price (p), and weight (W) that reflects

the user sensitivity to quality and price. We modified the original function in [SK98] and defined it using function (2).

$$U = W (q_i - MIN\_Q) / THD + (1 - W) (MAX\_P - p_i) / MAX\_P \quad (2)$$

Where:

- U is the user utility
- W is the weight reflecting the relative sensitivity to quality and price. It is negotiated and assigned when the user signs the SLA (see Appendix A) with the Internet Service Provider
- q_i is quality received in class i
- MIN_Q is the minimum quality that a user is willing to tolerate and willing to pay for (MIN_Q = 1 – THD)
- p_i is the price paid for class i
- MAX_P is the price paid for the highest class AF4 which his the maximum price

The original user utility function in [SK98] (function (2a)) does not consider the balance between the quality and price. That means the values are not normalized. So it is for analytical purpose. For our experimental purpose, we have to modify this function to make the values normalized.

$$U = (W(q - MIN\_Q) + (1 - W)(MAX\_P - p))\theta(MAX\_P - p)\theta(q-MIN\_Q) \quad (2a)$$

First of all, as function (2a) has a step function θ which is not used in our algorithm, we modified the function to function (2b).

$$U = W(q - MIN\_Q) + (1 - W)(MAX\_P - p) \quad (2b)$$

Second, function (2a) has two parts, one is for the quality and the other is for the price. If

one of the parts is much greater than the other one, the smaller one will be ignored. For example, the quality is a fraction (from 0 to 1) and the price is determined by 20 cents/byte. Then, no matter how big the weight of the quality is assigned, the quality part will be ignored as it is too small to affect the utility. So we divide the price by the maximum price (MAX_P) to make this part as a fraction too so that it has the same scale as the quality part (function (2c)).

$$U = W (q_i - MIN\_Q) + (1 - W) (MAX\_P - p_i) / MAX\_P \qquad (2c)$$

Third, we consider that the utility should evenly depend on the quality and the price regardless the weight. Let see the following numerical examples (Suppose THD = 0.2, MAX_P = 20 cents/byte, MIN_Q = 1 – THD = 0.8.):

For the price part,

$(MAX\_P - p_i) / MAX\_P = 0$ if $p_i = MAX\_P$

$(MAX\_P - p_i) / MAX\_P = 1$ if $p_i = 0$

Thus, the value range of the price part is from 0 to 1. For the quality part,

$q_i - MIN\_Q = 0$ if $q_i <= MIN\_Q$

$q_i - MIN\_Q = THD = 0.2$ if $q_i = 1$

The value range of the quality part is from 0 to 0.2. The quality has less influence on the utility function than the price. The price becomes the dominant fact when calculate the user utility. Moreover, it is the dominant fact to make the class switching decision as our class switching is based on the user utility. This is not correct for our algorithm.

To let the utility evenly depends on the quality and price, we divide the quality part by

THD to make this part has the same value rang (0 – 1) as the price. This results in our final utility function (function (2)).

$$U = W (q_i - MIN\_Q) / THD + (1 - W) (MAX\_P - p_i) / MAX\_P \quad (2)$$

The quality weight W indicates the user's sensitivity to the quality. There are several ways to set the quality weight W:

- If (qi < MIN_Q), then U = 0
- If (W = 0), then user chooses the service with the lowest price as long as the quality is no less than MIN_Q
- If (W = 1), then user chooses the service with the highest quality
- If (0 < W < 1), then user chooses the service based on the quality and price tradeoff

Chapter 6 shows different simulation results when choosing different values for W.

5.4 Algorithm

The Adaptive Class Switching Algorithm (ACSA) algorithm is explained in this subsection.

In addition to the variables explained above, there are other variables used in the algorithm which are listed in Table 5.1.

NUM_CLASS	Number of classes = 5
currClass	The class that our real-time multimedia data flow is running in
maxU	Maximum utility
switchClass	The class with the maxU

Table 5.1 Some variables used in the ACSA algorithm

When the sender receives a RTCP RR feedback report in class currClass in which the real-time multimedia flow is running, it will trigger the algorithm. The sender stores the new packet loss rate included in the RR into a table and uses all packet loss rates in the table belonging to all classes to calculate the user utilities in all classes (lines 3 – 11). If the packet loss rate of class i is greater than the threshold (THD), it means that the quality received in this class is less than the MIN_Q accepted. Then, the utility (U) for this class becomes zero (line 10). If the packet loss < THD, the algorithm will update the maximum utility (maxU) and the corresponding class (switchClass) to be switched to accordingly (line 7). After comparing all classes, if maxU is zero, that means that the quality of all the classes is bad and the Internet is congested. Thus, the application will switch to the BE class with no cost (line 13). When classes have higher utilities than zero, the application will switch back accordingly. At line 15, if maxU is greater than the utility of the current class, switching occurs (line 16). Otherwise, the application stays in current class (line 18).

```
1   Receive a new RR feedback in class currClass
2   maxU = 0                                                    // maximum utility
3   FOR each class i                                    // classes AF1..AF4 and BE
4       IF currPktLoss(i) < THD // in the most recent RR for class i
5       {       Compute the user utility (U(i)) of this class
6               IF U(i) > maxU
7                   Update maxU and the corresponding class (switchClass)
8       } ELSE
10              U(i) = 0
11  END FOR
12  IF maxU = 0
13      Application switches to BE
14  ELSE
15      IF currClass != switchClass
16          Application switches to switchClass
17      ELSE
18          Application stays in currClass
```

We probe all classes simultaneously to get the most recent Internet state by sender

sending the helper packets to these classes. The helper packets only contain RTP headers in order to minimize the additional network load.

By calculating the user utilities for all classes, the sender chooses the one with the highest user utility and switches to it directly. If several classes have the same highest utility, the sender chooses the class with the lowest numerical number. The class with the highest utility reflects the best quality and price tradeoff. Thus, the user is always willing to choose this class. Therefore, our algorithm provides users a good service with a good price when the Internet is in normal state. If all classes are congested, the algorithm chooses the BE class. Thus, there is no high cost for a bad service.

5.5 Complexity analysis

This part analyzes the time and space complexity for the ACSA algorithm. Time complexity determines the way in which the number of steps required by an algorithm varies with the input size of the problem it is solving. The space complexity determines the way in which the amount of storage space required by an algorithm varies with the input size of the problem it is solving.

This ACSA algorithm is composed by two main parts (see the algorithm above): user utility calculation (line 3 – line 12) and comparison (line 13 – line 19). Table 5.2 shows the time and space complexity for each part and the overall complexity.

Step	Line 1 - 2	Line 3 - 12	Line 13 - 19	Overall
Time complexity	O(1)	O(N)	O(1)	O(N)
Space complexity	O(N)	O(N)	O(1)	O(N)

Table 5.2 Complexity analysis

(Where N is the number of the DiffServ service classes. O(1) + O(N) = O(N))

The reason of why the space complexity of line 1 - 2 is O(N) is that we need to store the packet loss rates conveyed by the Receiver Reports into a table that is a one-dimension array (See detailed implementation in subsection 6.2). Thus, the overall complexity of the algorithm is O(N) where N is the number of the DiffServ service classes.

5.6 Conclusion

This chapter introduces the ACSA algorithm in details. It explains the goal, the assumption, and some main terminologies of the algorithm. It also explains the algorithm itself and analyzes its complexity.

Chapter 6 Simulation Environment

This chapter explains the simulation topology, implementation, different experimental scenarios, and the results.

6.1 Simulation topology

To evaluate the algorithm, we simulated, using NS-2 [NSHome], an Internet voice application that uses RTP/RTCP and that runs over a DiffServ network. The packet loss threshold (THD) for this application is set to 0.2 as this is the threshold that a human being can tolerate in this application. The network topology is shown in Figure 6.1.

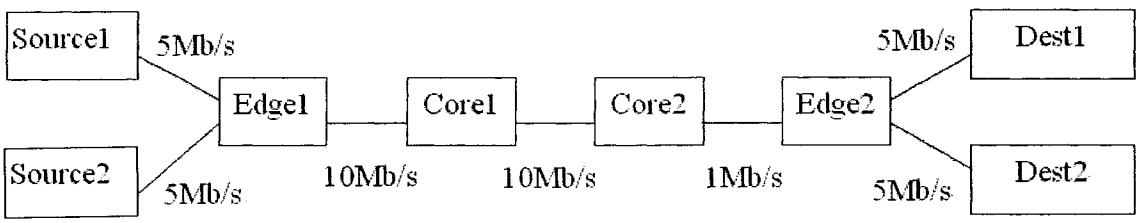

Figure 6.1 Simulation topology

Our real-time multimedia application runs in Source1 host. The sender in Source1 host sends a flow of size 512 bytes each at a high rate to Dest1. Other flows containing small packets that have RTP headers only are sent by Source1 into other classes to probe the current state in other classes. The packet size of a probing packet is 12 bytes. Source1 is a DiffServ capable node which can perform the DiffServ functions such as packet classification, marking, and shaping. Source2 is not a DiffServ node. An interfering application runs on Source2 host and sends to Dest2 UDP packets that overload the traffic in different DiffServ classes. Core1 and Core2 are DiffServ core routers and Edge1 and Edge2 are DiffServ edge routers. Dest1 and Dest2 hosts receive data from source1 and source2. The bandwidth of each link between a host and an edge router is set to 5Mb/s and the bandwidth of each link within the DiffServ domain (Edge1 to Edge2) is set to 10Mb/s except the bottleneck link between Core2 and Edge2 is set to 1Mb/s.

The real-time multimedia application path in the topology starts from Source1 – Edge1 – Core1 – Core2 – Edge2, until Dest1. Source1 sends real-time multimedia data and receives RTCP packets from Dest1. The packet size of the multimedia application is 512 bytes. Source1 also sends probing packets into other classes to probe the class states of these classes. The probing packets contain only RTP header and the size is 12 bytes which is much smaller than the multimedia application. We keep the probing packet small size to reduce the probing overhead as much as possible.

The interfering application path starts form Source2 – Edge1 – Core1 – Core2 – Edge2, until Dest2. Source2 sends UDP interfering traffic to Dest2.

6.2 Implementation

The implementation uses the Network Simulator (NS) 2.26 which we installed in Windows XP operating system. The programming languages of NS are Tcl and C++. C++ is used to implement new functions used in the simulation and Tcl is used as a front-end.

The implementation of the simulation contains 6 modules:

- Setting up and configuring the network topology
- Collecting the RRs and storing the current packet loss rates
- Calling the algorithm
- Class switching
- Recording
- Plotting the graphs

The following subsections explain each part in details.

6.2.1 Setting up and configuring the network topology

The network topology is very important to the simulation because it determines the packet forwarding behaviors and the service levels. In original DiffServ implementation in NS2, the flow is distinguished by the source and destination. So, between each source-destination pair there is only one flow. This is insufficient for our implementation because we need several flows transmitting between one source-destination pair. For example, Source1 sends multimedia data flow as well as probing flows to Dest1. To distinguish these flows, we use IPv6 packet in this implementation. In IPv6 data packet header, there is a flow label that is used to identify a flow [BM95]. In NS2, this functionality has been added by a DiffServ patch from [NSCon]. This patch provides the DiffServ classification and marking at edge node for flows distinguished by the source, destination, and flow id. We assign different flow id to different data flow and assign these flows to different classes.

Then we configure each link within the DiffServ domain. In our simulation, each link contains 5 DiffServ RED queues for 5 classes (BE, AF1 – AF4) respectively. We assign different parameters such as transmitting rate, burst size, and initial codepoint for different queues to perform different levels of service. For example, AF4 class has the highest transmitting rate and BE class has a rate of zero which means no minimum QoS guarantees at all. Thus, AF4 class should provide the best service and BE class should provide the lowest service under the same network condition.

Supporting the applications using RTP protocol over DiffServ network should configure the DiffServ queues for all RTP flows including the multimedia data flow and the probing flows. These flows join to the same multicast group. Thus, one RTCP RR contains information of all flows in the group. All RTP flows start at the beginning of the simulation.

6.2.2 Collecting the RRs and storing the current packet loss rates

The input of this module is the packet loss rate (loss) incurred in the current interval and the output is an updated table (currPktLoss).

When a RTCP RR comes, the packet loss rate incurred in the current interval contained in the RR is stored for further use. We use a 1-Dimension table (currPktLoss) to store the information. The table is updated when there is a new RR arriving so that it always keeps the most current packet loss information for all classes. Only the packet loss rate of the real-time multimedia data flow generates a call to the algorithm.

RRs summarize QoS of the transmission. When the sender sends the real-time multimedia flow, it receives RRs reporting feedback information about the quality received by the flow. After a class switching occurs, the first coming RR may not report the QoS of the new class but rather reports the QoS of the previous class, where the application (real-time multimedia flow) was running before switching, or a combination of the quality received in two classes. So this first RR does not reflect the QoS of the current class (i.e. the new class). To avoid network oscillations, we discard the first RR arriving after each switching and use the RR from the probing flow in the new class.

6.2.3 Calling the algorithm

When a RR belonging to the real-time multimedia flow is received by the sender, the algorithm is called. The utility of all classes will be compared, and the real-time multimedia flow will switch to the class with the highest utility. So, the only delay that occurs for switching to another class will be the time between issuing the RR by the receiver until it is received by the sender plus the utilities' comparison time at the sender which is very minimal. The input of this module is the class name of which the application is running. This module decides whether to call the class switching module or

not.

The C++ function getCodePoint is added for retrieving the current class where the application is running. The module uses the packet loss rates stored in currPktLoss table to calculate the user utilities for all classes and updates the maximum utility and the corresponding class to be switched to.

Theoretically, MAX_P is the highest price paid for the service and it should be the price for class AF4. In the implementation, we have to set it greater than the price paid for class AF4, otherwise, the price part of the utility for class AF4 is always 0. Moreover, when the quality weigh W = 0, the utility for class AF4 is always 0. That means, if MAX_P = p_4, (1 - W) (MAX_P - p_4) / MAX_P = 0. Then

U = W (q_4 - MIN_Q)/THD + (1 - W) (MAX_P - p_4) / MAX_P = W (q_4 - MIN_Q)/THD

If W = 0, then U = W (q_4 - MIN_Q)/THD = 0. This is not true for AF4 class. Thus, we have MAX_P > p_4.

The current utility for each class stores in a 1-D table U. The maximum utility maxU and the corresponding class switchClass are updated while computing the utilities. The table U, maxU, and switchClass are reset each time the algorithm module is called.

After the algorithm computes the maxU and knows the class to switch to (switchClass), it compares maxU with the utility of the current class (currClass) where the application is running (see subsection 5.5). If the condition is satisfied and the currClass != switchClass, switching occurs.

6.2.4 Class switching

Class switching means resetting the transmission parameters for the multimedia

application flow. The input of the module is the class to switch to and there is no output.

To let the real-time multimedia flow reacts to the network state as fast as possible, the parameters should be adjusted in run-time instead of stopping the transmission, changing parameters, and restarting the transmission. However, NS2 does not support dynamically adjusting the parameters for DiffServ network. We have added some functions and imported other functions from the contributed code [NSCon]. There are two functions for changing the parameters dynamically and they are implemented in C++:

- updateCodePoint is used to reassign the service class for the flow by updating the DiffServ CodePoint (DSCP) stored in IP packet header. The DSCP indicates the service class in DiffServ network.
- changeMeterParms is used to change the transmission parameters for the flow after the flow switches to a new class. These parameters define the QoS provided by the service class.

6.2.5 Recording

This module is used to record the history of the packet loss rates, from the beginning of the transmission, for all probing flows and the real-time multimedia flow. Also, it records all classes where real-time multimedia flow was running throughout the life time of the session. The procedure call is generated every 0.5 second. It writes the packet loss rates into files for displaying the graphs.

6.2.6 Plotting the graphs

This module is the final procedure to stop the transmission, close the recording files, and plot the graphs according to the data stored in the recording files. These graphs show the history of the packet loss of all classes. Also, to make the comparison clearer, we generate the packet loss and service class for the real-time application separately. These

graphs show the goals and the problems solved by our algorithm.

6.3 Simulation experiments

Initially, our real-time multimedia data flow runs in BE class as this service has no cost. In the next subsection, we present different simulation scenarios with different values of the quality weight (W) to show how the class switching algorithm reflects the quality and price tradeoff. The total simulation time is 30s. The data recording time period is 0.5.

The simulation experiments test and compare the class switching behaviors under the following conditions (scenarios):

- Application without implementing the adaptive class switching algorithm (ACSA)
- Application implementing ACSA
  - Quality weight W = 0 - Switching based on the lowest price
  - Quality weight W = 1 - Switching based on the highest quality
  - Quality weight W = 0.5 - Switching based on the tradeoff between quality and price

We choose different W to test how this parameter impacts the switching behaviors. W=0 expresses the extreme price sensitivity and W=1 expresses extreme quality sensitivity. While W=0.5 gives the quality and price the same weight to the user utility, i.e., the user have equal sensitivity to the price and the quality. Setting W to these three values can represent all features of our algorithm. When W=0.5, we simulate 2 Internet states to show that our algorithm has different switching results than the other related work. These states are the higher class in the Internet is overloaded, and the Internet is congested.

6.3.1 Application without implementing the Adaptive Class Switching Algorithm

In this scenario, the interfering traffic flows run in BE and AF1 classes. These flows start at time 5s and stop at 15s.

Without implementing the ACSA algorithm, the application (real-time multimedia flow) stays in the same class as when it is initialized (in BE class). If there is overload traffic in the class, the application suffers high packet loss rate until the interfering traffic stops (See Figure 6.2).

As we talked in subsection 6.2.2, the feedback information conveyed in RRTCP RRs summarizes QoS of the transmission. It reports the transmission quality received before it is sent out. Thus, when the interfering flows start at time 5s, the feedback returns to the sender at time 6.5s. The feedback shows that the application (real-time multimedia flow) starts to suffer a packet loss in BE and its QoS has no improvement until time 17.5 because the interfering traffic flows stop at time 15s (same reason as why the interfering flows start at time 5s and the packet loss occurs at time 6.5s). During this time period, the application suffers high packet loss rates and most of them are greater than the THD. Meanwhile, the QoS in other classes are much better (packet loss rate is zero) but the resources are wasted because the application is always running in the BE class and cannot switch to other classes even if these class resources are idle. From the following experiments, we will see that our algorithm solves this problem by switching to other class to utilize the network services and improve the QoS of the application.

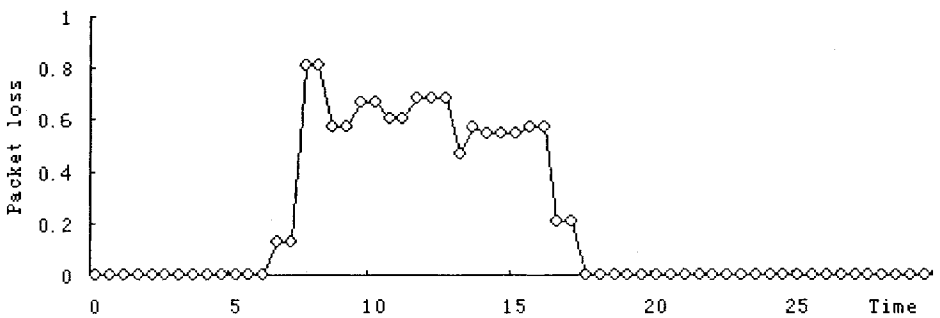

Figure 6.2 Packet loss of the real-time multimedia flow versus time

(No ACSA implementing)

6.3.2 Switching based on the lowest price

When the user is very sensitive to the price, he will choose the quality weight (W) to have a smaller value, even set it to 0. In this case, the user utility will depend mainly on the price part, i.e. $U = (1 - W)(MAX\_P - p_i) / MAX\_P$

In our ACSA algorithm, we assumed that the classes are ordered in terms of price. A class with a higher numerical number has a higher price than a lower numerical number class (see subsection 5.1). In this case, the class switching is based on the lowest price as long as the current packet loss rate is less than the threshold. Here we set the packet loss threshold THD = 0.2 because this is a threshold that human beings can tolerate in a voice application. The real-time multimedia flow will stay in the lowest class (lowest numerical number class with lowest price) which can guarantee the QoS requirement (i.e., packet loss < THD). Figure 6.3 shows the results. In the following subsections, we use figures 6.Na, 6.Nb, and 6.Nc, where N=3..7. Figures Na show the packet loss for all classes considered in our algorithm (BE, AF1..AF4). If the packet loss of a class is constantly zero during the simulation time, then it will not show in the figure. Figures 6.Nb show the packet loss incurred by our real-time multimedia application flow. Figures 6.Nc show the class where the real-time multimedia flow was running during the simulation.

In this scenario, the interfering flows in BE and AF1 classes start at time 5s and stop at 15s.

- At time 7.5s, the feedback shows that the quality in BE degrades (Fig. 6.3a). The application (real-time multimedia flow) switches to AF1 (Fig. 6.3c). The packet loss of the application reduces to zero soon (Fig. 6.3b) because it is using the resource in

AF1 class now and the packet loss rate in this class is zero at this time.

- After time 8.5s, the packet loss in AF1 (Fig. 6.3a, Fig. 6.3b) increases because the application increases the traffic in AF1. Before it reaches THD, the application stays in AF1 because AF1 class is the lowest class that satisfies the QoS requirement of the application (packet loss < THD). The utility table (Table 6.1) shows that during time 7.5s to 15s, the utility of AF1 class is the highest utility.

- At time 15s, the packet loss of AF1 is over THD because that the resource assigned to AF1 class is used out (queuing buffer is full). Then the utility for AF1 reduces to 0 and the application switches to AF2 right away.

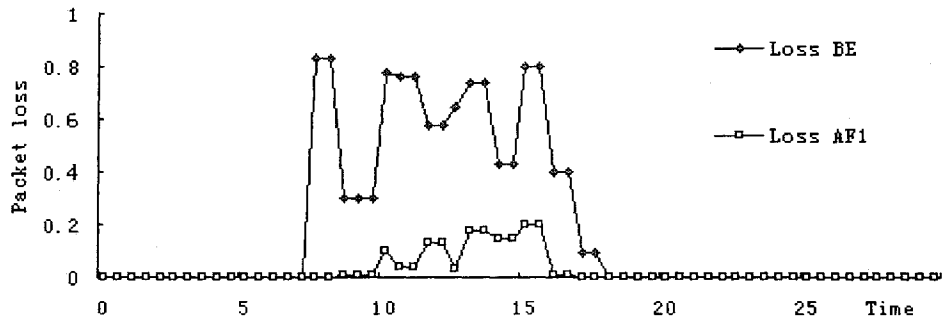

Figure 6.3a Packet loss in BE and AF1 versus time.

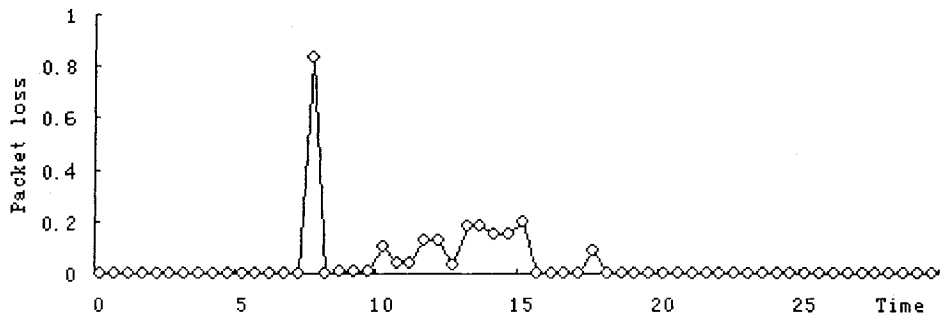

Figure 6.3b Packet loss of the real-time multimedia flow versus time

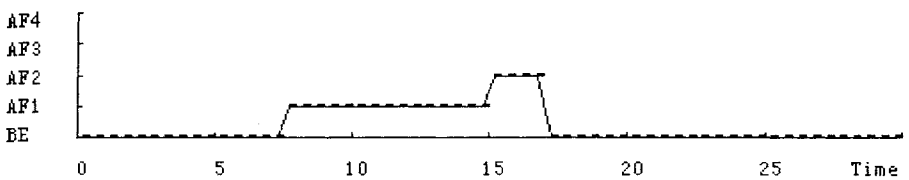

Figure 6.3c The class where the real-time multimedia flow is running versus time

Figure 6.3 Packet loss and class switching based on the lowest price (W = 0)

Time	BE	AF1	AF2	AF3	AF4
7.5	0	0.83	0.67	0.5	0.33
15.0	0	0	0.67	0.5	0.33
17.0	1.0	0.83	0.67	0.5	0.33

Table 6.1 User utilities – switching based on the lowest price (W=0)

- After the interfering flows stop, the packet loss in BE as well as AF1 decrease and the application switches back to BE at time 17s.

The results show that at anytime, the real-time multimedia flow seeks to stay running in the lowest numerical number class which meets the QoS requirement, i.e., packet loss < THD (Fig. 6.3, Table 6.1).

6.3.3 Switching based on the highest quality

When the user is very sensitive to the QoS of the application, he will choose the quality weight (W) to have a higher value, even set it to 1.0. In this case, the user utility depends mainly on the quality part. i.e. U = W (qi - MIN_Q) / THD.

Switching will be based on the lowest numerical number class offering the best QoS regardless of the class price. In this scenario, there are three interfering traffic in BE, AF1, and AF2 classes. Interfering flows in BE and AF1 start at time 5s and stop at 15s. The interfering flow in AF2 starts at time10s and stops at 20s. The purpose of using the short interfering in AF2 is for the comparison to the next scenario – Switching based on both quality and price. Figure 6.4 shows the results.

- At time 7.5s, when the interfering flows cause the quality degradation in the BE class (Fig. 6.4a), the application (real-time multimedia flow) switches to AF1 (Fig. 6.4c) and its packet loss reduces to zero (Fig. 6.4b) (same reason as in subsection 6.3.2).
- When packet loss in AF1 increases at time 10s, the application switches to AF2 (Fig. 6.4c) because it seeks the lowest class offering the best QoS which is AF2 at this time. The packet loss reduces to zero soon (Fig. 6.4b).

- The application flow and the interfering flow overload AF2. At time 13s, there is a very small packet loss (0.01) in AF2 which is too small to be shown in Fig. 6.4b. Because the switching is based on the best quality in this case, the application seeks to switch to the lowest class with the best quality no matter how small the packet loss is in the current class. So, it switches to AF3 class and stays there until the packet loss of AF2 becomes zero.

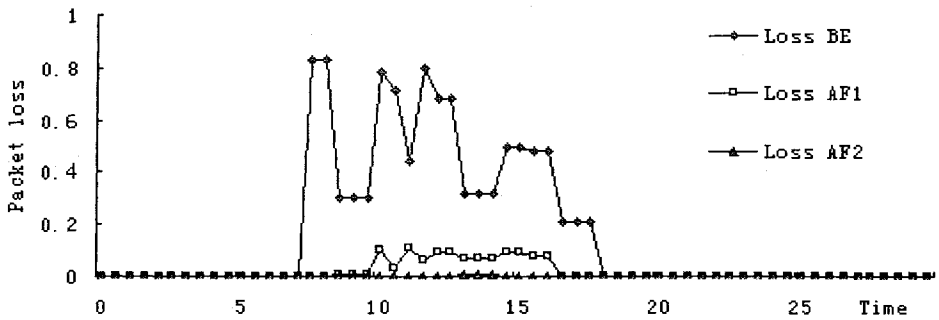

Figure 6.4a Packet loss in BE, AF1, and AF2 versus time

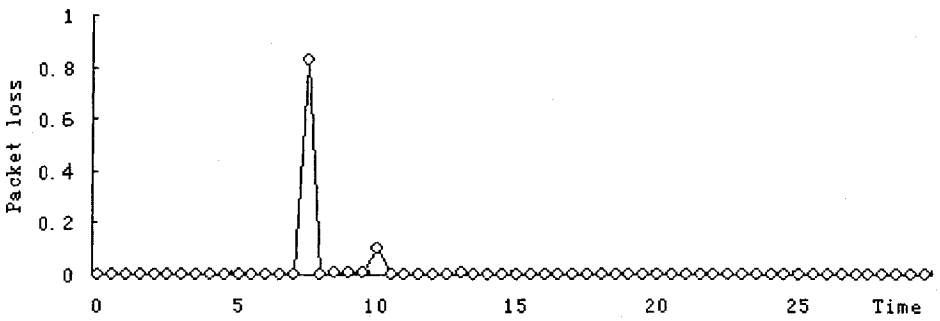

Figure 6.4b Packet loss of the real-time multimedia flow versus time

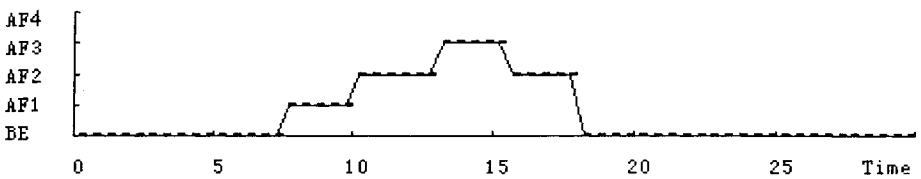

Figure 6.4c The class where the real-time multimedia flow is running versus time

Figure 6.4 Packet loss and class switching based on the highest quality (W =1)

Time	BE	AF1	AF2	AF3	AF4
7.5	0	1.0	1.0	1.0	1.0
10.0	0	0.5	1.0	1.0	1.0
13.0	0	0.65	0.95	1.0	1.0
15.5	0	0.60	1.0	1.0	1.0

18.0	1.0	1.0	1.0	1.0	1.0

Table 6.2 User utilities – switching based on the highest quality (W=1)

- At time 15.5s, the application switches to AF2 because AF2 is the lowest class which offers best QoS (see table 6.2).
- After the interfering flows in BE and AF1 class stop, the packet loss in all classes decrease to zero (Fig. 6.4a) and the application switches to BE at time 18s (Fig. 6.4c).

The results show that at anytime, the real-time multimedia flow tries to stay in the class that is the lowest class providing the highest utility (in this case, it means the highest quality or lowest packet loss (Fig. 6.4, Table 6.2)).

6.3.4 Switching based on both quality and price

Usually, users are interested in receiving a good service with a good price. Without loss of generality, we set weight W= 0.5. The results show how both the price and the quality are taken into account and how they influence the switching results.

The initial state of this scenario is the same as the one described in subsection 6.3.3 except that the weight is 0.5. Interfering flows in BE and AF1 start at time 5s and stop at 15s. The interfering flow in AF2 starts at time 10s and stops at 20s.

- At time 7.5s, when the interfering flows cause the quality degradation in the BE class (Fig. 6.5a), the application (real-time multimedia flow) switches to AF1 (Fig. 6.5c) and its packet loss reduces to zero (Fig. 6.5b).
- At time 10s, the packet loss in AF1 is 0.1 which is less than THD (Fig. 6.5a, Fig. 6.5b). Because the application considers the price and quality together, it switches to AF2 (Fig. 6.5c) which is the class with the highest utility (Table 6.3). Meanwhile, the interfering flow in AF2 starts. The interfering here is used to degrades the quality of this class to a certain extend.

- At time 13s, the packet loss in AF2 is 0.01 (Fig. 6.5a). Again, because the application considers the quality as well as the price, AF2 is still the class with the highest user utility and the application stays running here.
- After the interfering flows in BE and AF1 stop, the application switches to AF1 at 17.5s, then to BE at 19.5s.

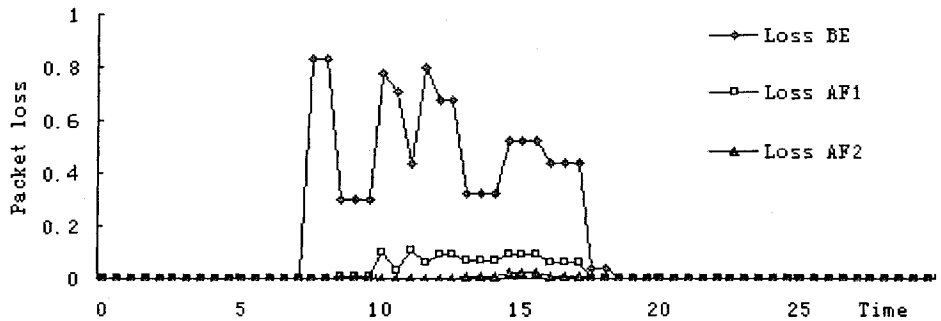

Figure 6.5a Packet loss in BE, AF1, and AF2 versus time

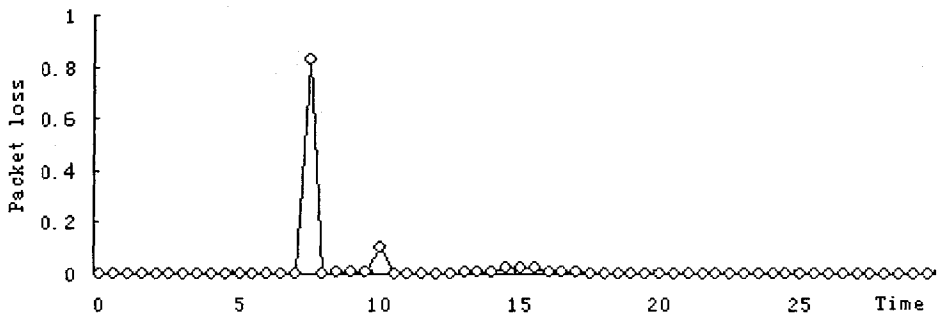

Figure 6.5b Packet loss of the real-time multimedia flow versus time

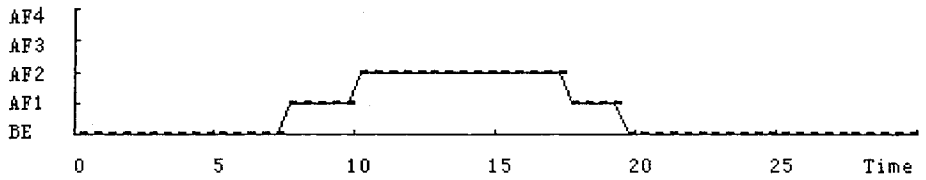

Figure 6.5c The class where the real-time multimedia flow is running versus time

Figure 6.5 Packet loss and class switching based on both quality and price (W=0.5)

Time	BE	AF1	AF2	AF3	AF4
7.5	0	0.92	0.83	0.75	0.67
10.0	0	0.67	0.83	0.75	0.67
17.5	0.9	0.92	0.83	0.75	0.67
19.5	1.0	0.92	0.83	0.75	0.67

Table 6.3 User utilities – switching based on both quality and price (W=0.5)

Comparing this result (Fig. 6.5c) to the one in subsection 6.3.2 (W = 0, switching based on the lowest price) between time 7.5s to 15s (Fig. 6.3c), we see different switching results. In subsection 6.3.2, switching is based on the lowest price class which can provide packet loss < THD. During the period between time 7.5s and 15s, although the application suffers a packet loss in AF1, the application stays running in AF1 (Fig. 6.3b, Fig. 6.3c) as long as the packet loss of AF1 is less than THD. When W=0.5, the switching is based on both quality and price. So when the utility of AF1 is less than the utility in AF2 at time 10s, the application switches to AF2 (Fig. 6.5c).

Comparing this result (Fig. 6.5c) to the one in subsection 6.3.3 (W = 1, switching based on the highest quality) between time 10s to 15s (Fig. 6.4c), the different results show how the weight W influences the switching decision. In subsection 6.3.3, switching is based on the lowest class offering the best QoS. At time 13s, the packet loss in AF2 is 0.01 in both cases. When W=1, the application switches to AF3 (Fig. 6.4c) because it takes the quality only into account. When W = 0.5, the application considers both quality and price and AF2 is still the best choice. The application stays in AF2 (Fig. 6.5c) with no change.

These results show that when the weight is between 0 - 1 (0<W<1), the algorithm can control the tradeoff between price and quality and switching is based on the highest utility.

6.3.5 Switching when higher classes are overloaded

Each AF subclass in a DiffServ node is allocated a certain amount of resources for minimum QoS guarantees and is priced based on the QoS guarantees. At the beginning, we assume that service classes in the DiffServ architecture are ordered in terms of price. In a normal network state, a higher price class should provide better QoS than that provided by a lower price class. However, when some of the classes in the Internet are overloaded, DiffServ may not be able to guarantee the QoS for the application. In the

worst case, the quality among classes may be disordered – higher more expensive classes provide less quality than lower cheaper classes. Previous work did not handle this case [NV00, DR01, SB02], i.e. when the QoS requirement is violated in a class, the application switches directly to a higher class. But using our algorithm, we probe all classes simultaneously and the switching is based on the highest utility. Application will switch to the class with the highest utility directly and the new class may be lower than the current class. Figure 6.6 shows the results of this scenario.

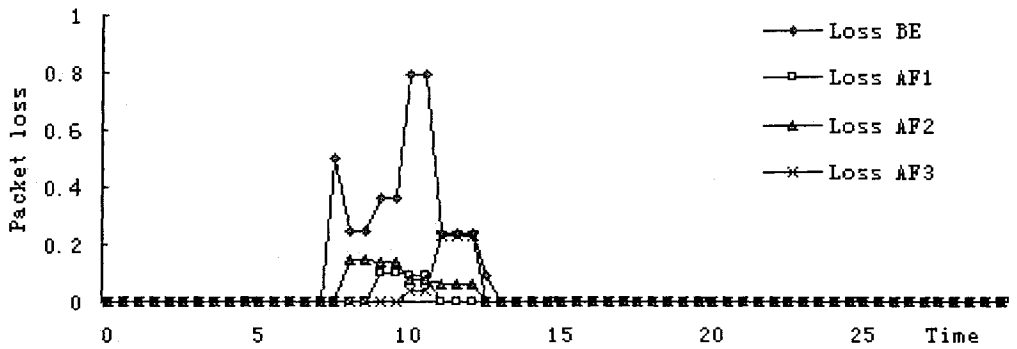

Figure 6.6a Packet loss in BE, AF1, AF2, and AF3 versus time

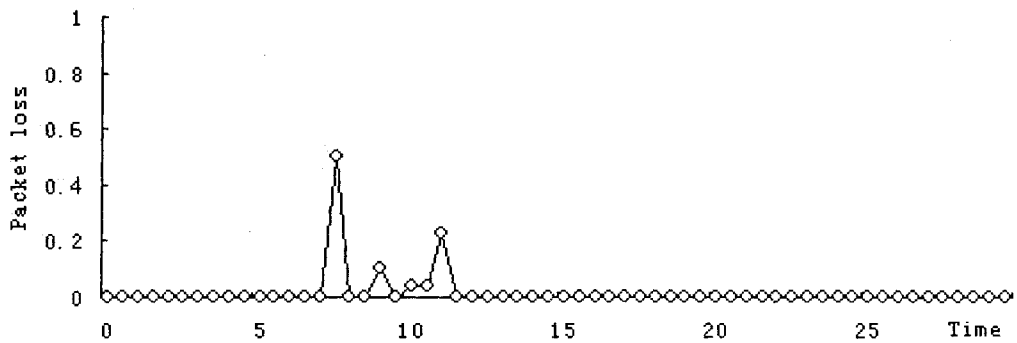

Figure 6.6b Packet loss of the real-time multimedia flow versus time

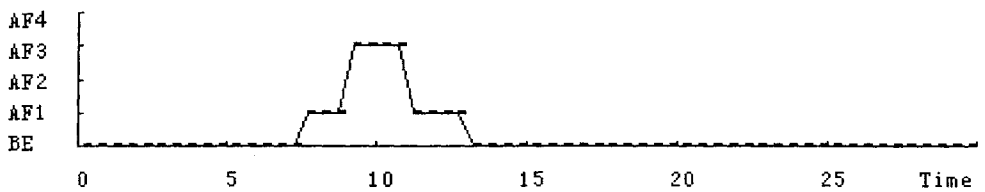

Figure 6.6c The class where the real-time multimedia flow is running versus time

Figure 6.6 Packet loss and class switching when higher classes are overloaded (W=0.5)

Time	BE	AF1	AF2	AF3	AF4
7.5	0	0.92	0.83	0.75	0.67

9.0	0	0.67	0.48	0.75	0.67
11.0	0	0.92	0.68	0	0.67
13.0	1.0	0.92	0.83	0.75	0.67

Table 6.4 User utilities –switching when higher classes are overloaded (W=0.5)

In this scenario, class switching is based on both quality and price (W = 0.5). At the beginning the application (real-time multimedia flow) runs in the BE class. Interfering flows in BE and AF2 classes start at time 5s and stop at 10s. The interfering flow in AF3 class starts at time 7s and stops at 15s and it is used to overload the higher class (AF3).

- When the application experiences high packet loss in the BE class at 7.5s (Fig. 6.6a, Fig. 6.6b), it switches to AF1 class (Fig. 6.6c). Before the application makes the next switching decision, it has the choices of switching to higher or lower classes.
- The application decreases the quality in AF1, so at time 9s, it switches to AF3 class which has the highest utility (Table 6.4).
- At time 11s, the application suffers a packet loss over THD in AF3 because of the interfering flow and itself. At the same time, the packet loss in AF1 and AF2 are below THD. So, instead of switching to a higher class, our application switches to AF1 which has the highest utility (Table 6.4).
- After the interfering flows in BE and AF2 stop, the packet loss in BE, AF2, and AF3 reduce to 0 at time 13s. The application switches to BE.

These results show that when higher classes are overloaded, the QoS in higher classes may be worse than lower classes and with our algorithm, the application can switch to the most suitable class directly.

6.3.6 Switching when all classes are congested

The interesting results using ACSA algorithm show in Figure 6.7 when there are interfering flows in all classes so that the network gets fully congested. According to our

assumption, if the quality in all classes cannot be guaranteed so that the highest utility is zero, the application will use the BE service with no payment. This result is quite different than results in other previous work. In [CZ00], if the highest utility is zero, the application will not use any of the classes, i.e. the application will be dropped. In [NV00], if the application cannot receive the required QoS, it will remain in its maximum class that it is willing to pay. In [DR01, SB02], at this time, the application stays in the highest class unsatisfied.

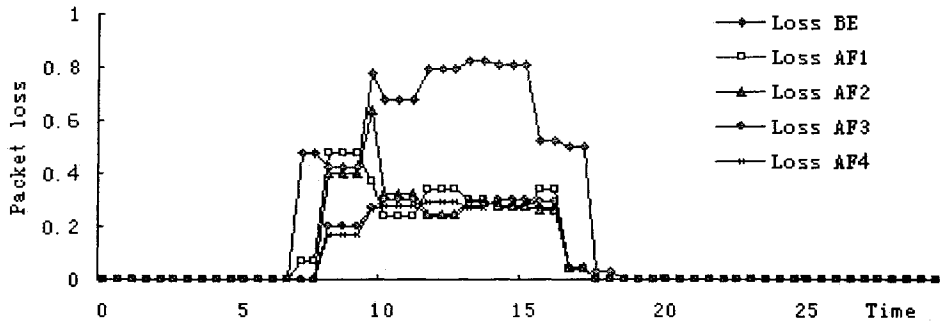

Figure 6.7a Packet loss in all classes versus time

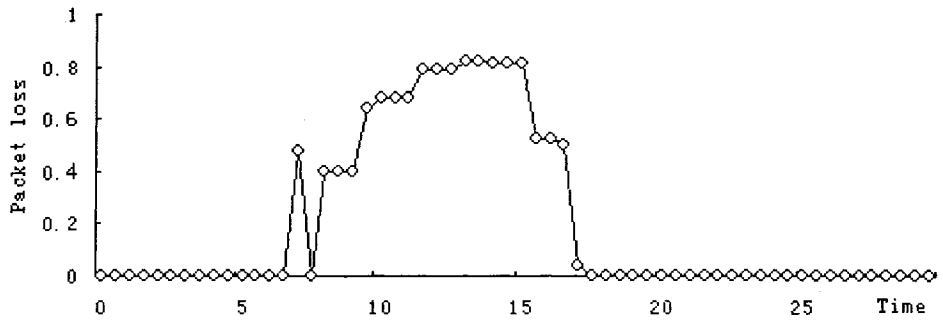

Figure 6.7b Packet loss of the real-time multimedia flow versus time

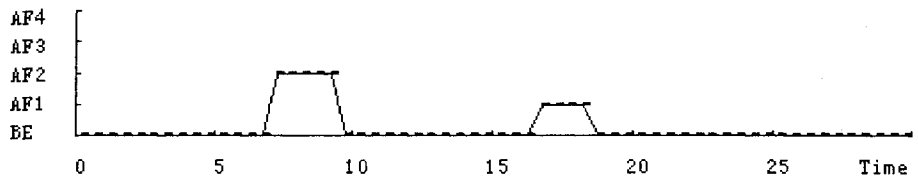

Figure 6.7c The class where the real-time multimedia flow is running versus time

Figure 6.7 Packet loss and class switching when all classes are congested (W=0.5)

Time	BE	AF1	AF2	AF3	AF4
7.0	0	0.68	0.83	0.75	0.67
9.5	0	0.78	0.67	0.62	0.53

18.5	1.0	0.92	0.83	0.75	0.67

Table 6.5 User utilities – switching when all classes are congested (W=0.5)

In this case, the quality weight (W) is 0.5 so the class switching is based on both quality and price. There are 2 interfering flows in each class. At time 5s, two interfering flows start in BE and AF1, one in each class. Then, the other 8 flows start at time 7s. All interfering flows stop at 15s. These interfering flows are used to overload the network.

- The interfering flows starting at time 5s in BE and AF1 degrade the quality in these two classes. Therefore, at time 7s, the application (real-time multimedia flow) switches to AF2 class (Fig. 6.7a, 6.7b). Meanwhile, the other interfering flows start and overload the network.
- At time 9.5, the packet loss rates in all classes are over THD (Fig. 6.7a) and all utilities are zero (Table 6.5). This indicates that there is congestion in the network. Because the quality in all classes cannot be guaranteed, why pay money for the bad quality? Our application switches to BE class right away (Fig. 6.7c).
- The application stays in class BE until the packet loss in AF1 is below the THD and the highest utility is greater than zero at time 16.5s after all interfering flows stop. Then it switches back to BE.

These results show that at time of congestion, the application will use BE class with no payment for the bad services.

6.4 Conclusion

This subsection explains the simulation topology, implementation, different scenarios, and the results. The simulation results show that the ACSA algorithm will allow the real-time multimedia data flow to switch fast to the class with the highest utility. The followings are the results of different experimental scenarios:

- When the switching is based on price only (W = 0), the real-time multimedia flow chooses the lowest cheapest class (i.e., class with the lowest numerical number) which meets the QoS requirement, i.e. packet loss < THD.
- When the switching is based on QoS only (W = 1), the real-time multimedia flow chooses the lowest class with the best quality, i.e. the lowest packet loss incurred.
- When the switching is based on both QoS and price (0 < W < 1), the real-time multimedia flow chooses the lowest class which has the highest utility.
- When the higher classes are overloaded in the network, the real-time multimedia flow may choose a lower class which offers the highest utility.

- When the network is congested and the QoS cannot be satisfied in all classes, the real-time multimedia flow stays in BE class with no payment.

Chapter 7 Conclusion and Future Work

This chapter presents the conclusion and future research directions for the ACSA algorithm.

This thesis proposes an Adaptive Class Switching Algorithm (ACSA) that provides dynamic QoS control for real-time multimedia applications in a DiffServ environment. Our algorithm focuses particularly on DiffServ EF and AF services.

The DiffServ EF and AF services provide class-based QoS and these services are associated with the cost. Usually, the high-cost class provides high QoS guarantees. However, when the Internet is congested, DiffServ may not be able to guarantee the QoS for the application. Thus, the QoS may not reflect the price paid for the service. How to achieve a good price and quality tradeoff even at times of congestion is the problem solved in this thesis.

The ACSA algorithm combines the techniques of RTP protocol, DiffServ, and Adaptation together and works in a general DiffServ environment even at times of congestion. It also takes both QoS and price into account to provide users a good QoS with a good price. The user utility is a function of quality, price, and the weight of quality and it reflects the quality and price tradeoff. The quality is measured using the fraction of packet loss (i.e., the packet loss incurred in the current interval) carried by RTCP Receiver Reports (RRs). The adaptation is achieved by dynamically selecting the most suitable class based on the highest user utility which reflects the best quality and price tradeoff.

The simulation results show that the ACSA algorithm allows the real-time multimedia data flow to switch fast to the class with the highest utility. It reflects the best quality and price tradeoff. It always seeks to find a class with the highest user utility except when the

Internet is congested and the required QoS in all classes can not be satisfied. If this happens, the real-time multimedia flow chooses Best-Effort class with no cost. The followings are detailed results in different network scenarios:

- When the switching is based on price only (weight of quality W = 0), the real-time multimedia flow chooses the lowest cheapest class (i.e., class with the lowest numerical number) which meets the QoS requirement, i.e. packet loss < THD.
- When the switching is based on QoS only (W = 1), the real-time multimedia flow chooses the lowest class with the best quality, i.e. the lowest packet loss incurred.
- When the switching is based on both QoS and price (0 < W < 1), the real-time multimedia flow chooses the lowest class which has the highest utility.
- When the higher classes are overloaded in the network, the real-time multimedia flow may choose a lower class which offers the highest utility.
- When the network is congested and the QoS cannot be satisfied in all classes, the real-time multimedia flow stays in BE class with no payment.

Some extensive work needs to be done for future improvement.

- First, in our algorithm, the packet loss threshold THD is negotiated by the user and the Internet Service Provider (ISP) when signing the Service Level Agreement (SLA). In reality, it is more flexible if the THD for different applications varies. For example, Video conference and Internet telephony applications may require lower packet loss threshold than Internet gaming. Also, as the network state varies, the fixed THD may not be suitable for the network state throughout the duration of the RTP session. This may cause oscillations or network instability. In the future, the algorithm should be able to dynamically assign THD during the lifetime of the real-time multimedia application.

To dynamically calculate the THD, we consider using the exponential weighted average function to estimate the packet loss carried in the next RR. The THD that is going to be used when the next RR arrives is derived from the estimated packet loss. The THD can be changed within a range, e.g., a range $0.1 \leq THD \leq 0.2$ is for an Internet telephony application (Here we set the lower bound to 0.1 to avoid the network instability and set the upper bound to 0.2 as this is a voice quality threshold that human beings can tolerate). In this way, the THD can reflect the recent network state. Moreover, we will put more weight on the current packet loss so that the THD reflects the current network state closely.

- Second, we have considered only one receiver in the simulation. If there are several receivers with heterogeneous capabilities in the same RTP session, e.g., one is connected to the network through a modem and another through the high speed link, we need a mechanism to calculate the average packet loss of each class and use this average to calculate the user utility of the class.

- Third, from the user's perspective, the user perceived quality of the real-time multimedia application is not the same as the quality measured in packet loss. In fact, the principle of diminishing returns is applicable in this case. For example, when the quality of transmission summarized in the RRs shows there is a slight quality increase from the minimum quality, the user perceives a proportional utility for this improvement in quality. Further improvement in quality (measured by decrease in packet loss) has less effect on the user's perceived quality. The relationship between the quality summarized in the RRs and the quality perceived by the user can be divided into three regions:

  1. Region1 (Linear Region): In this region, improvement in quality is proportional to user's perceived quality.

  2. Region 2: This is the region for diminishing returns, i.e. improvement in quality

(measured by decrease in packet loss) has less effect on the user's perceived quality.

3. Region 3: whatever increase in quality is not perceived by the user.

So, future work involves adjusting the quality component of the utility calculation to include another function, e.g. the hyperbolic tangent function Tanh(βx), where β is a constant and x is [(q-MIN_Q)/THD]. Figure 7.1 shows the plot of this function with β = 3 and -1 < [(q-MIN_Q)/THD] < 1. Assume x = [(q-MIN_Q)/THD] and y = user's perceived quality.

Let us only consider the part with 0 < x < 1.

a. When 0 < x ≤ 0.3, there is a linear increase for y (0 < y ≤ 0.74). This part corresponds to region 1.

b. When 0.3 < x ≤ 1, y has a smooth increase from 0.74 to 1 and this part corresponds region 2.

c. When x approaches 1, y approaches the user's perceived normalized quality 1, and this part corresponds to region 3.

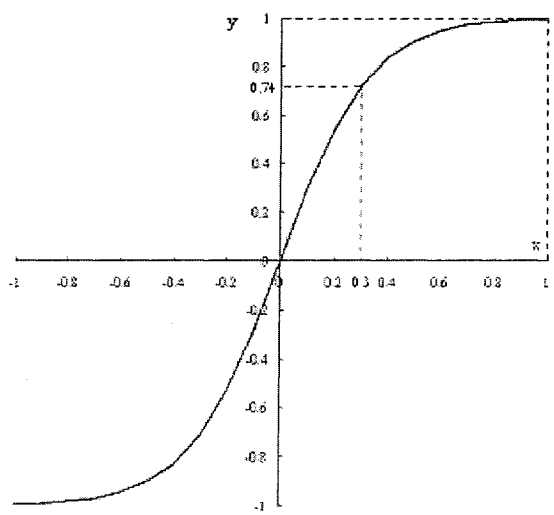

Figure 7.1 Tanh(βx), where β=3, x = [(q-MIN_Q)/THD], y=user's perceived quality

References

[BB98a] Y. Bernet, S. Blake, et al, "A Framework for Differentiated Services", Internet Draft <draft-ietf-DiffServ-framework-01.txt>, October, 1998

[BB98b] S. Blake, D. Black, et al, "An Architecture for Differentiated Services," IETF RFC-2475, December. 1998.

[BCS94] R. Braden, D. Clark, S. Shenker, "Integrated Services in the Internet Architecture: an Overview", RFC1633, June 1994

[BE00] L. Breslau, E. Knightly, et al, "Endpoint Admission Control: Architectural Issues and Performance", ACM SIGCOMM 2000

[Bha02] B. Bhargava, "Guest Editorial: Quality of Service in Multimedia Networks", Multimedia Tools and Applications an International Journal, Vol. 17, No. 2-3, July-August 2002

[BM95] S. Bradner, A. Mankin, "The Recommendation for the IP Next Generation Protocol", RFC 1752, 1995

[BZ97] R. Braden, L. Zhang, et al, "Resource ReSerVation Protocol (RSVP)", RFC 2205, September 1997

[CB95] K. Claffy, H. Braun, et al, "A parameterizable methodology for Internet traffic flow profiling", IEEE Journal on Selected Areas in Communications, Vol. 13, Issue 8, October 1995

[CSZ92] D. Clark, S. Shenker, L. Zhang, "Supporting real-time applications in an Integrated Services Packet Network: architecture and mechanism", Proc. on Communications architectures & protocols, pp.14-26, August, 1992

[CZ00] H. Che, S. Zheng, et al, "A model analysis of pricing and link bandwidth allocation in a multiple class-of-service network", Proc. of the 9th International Conference on Computer Communications and Networks, October 2000

[DR99] C. Dovrolis and P. Ramanathan, "A Case for Relative Differentiated Services and Proportional Differentiation Model," IEEE Network, vol. 13, pp.2-10, September-October 1999

[DR01] C. Dovrolis and P. Ramanathan, "Dynamic class selection: From relative differentiation to absolute QoS", Proc. of the Ninth International Conference on Network Protocols, November 2001.

[EL01] R. El-Marakby, "NS Implementation of RTP/RTCP", Proc. of the 1st IEEE International Symposium on Signal Processing and Information Technology, Cairo, Egypt, pp. 380-384, December 2001

[EK02] A. Elmaghraby, A. Kumar, et al, "Bandwidth allocation in a dynamic environment using a variable pricing policy", Proc. of 7th International Symposium on Computers and Communications, July 2002

[FJ93] S. Floyd, V. Jacobson, "Random early detection gateways for congestion avoidance", IEEE/ACM Transactions on Networking (TON), vol.1, No. 4, pp. 397-413, August 1993

[HB99] J. Heinanen, F. Baker, et al, "Assured Forwarding PHB Group," RFC 2597, IETF, June 1999

[HG99] J. Heinanen, R. Guerin, "A Single Rate Three Color Marker", RFC 2697, September 1999

[JN99] V. Jacobson, K. Nichols, et al, "An Expedited Forwarding PHB," RFC 2598, 1999

[Kil99] K. Kilkki, "Differentiated Services for the Internet", Macmillan Technology Series, 1999

[KR03] J. F. Kurose, K. W. Ross, "Computer Networking A Top-Down Approach Featuring the Internet", second edition, Addison Wesley, 2003

[LH03] M. Li, D. Hoang, et al, "Fair intelligent admission control over DiffServ network", the 11th IEEE International Conference on Networks, September-October 2003

[Liu00] C. Liu, "Multimedia Over IP: RSVP, RTP, RTCP, RTSP", 200 , http://www.cse.ohio-state.edu/~jain/cis788-97/ip_multimedia/index.htm

[NB98] K. Nichols, S. Blake, et al, "Definition of the Differentiated Services Field in the IPv4 and IPv6 Headers," RFC 2474, December 1998

[NShome] http://www.isi.edu/nsnam/ns/

[NSCon] http://www.isi.edu/nsnam/ns/ns-contributed.html

[NV00] T. Nandagopal, V. Venkitaraman, et al, "Delay differentiation and adaptation in core stateless networks", Proc. of the IEEE INFOCOM 2000 (19th Annual Joint Conference of the IEEE Computer and Communications Societies), vol. 2, pp. 26-30 March 2000

[Pos80] J. Postel, "User Datagram Protocol", RFC 768, August 1980

[SB02] M. Scheidegger, T. Braun, "An Adaptive IP Telephony Application over Differentiated Services", Proc. of the 21st IEEE International Conference on Performance, Computing, and Communications, April 2002

[SC03] H. Schulzrinne, S. Casner, et al, "RTP: A Transport Protocol for Real-Time Applications", RFC 3550, July 2003

[SC98] H. Schulzrinne, Columbia U., et al, "Real Time Streaming Protocol (RTSP)", RFC 2326, April 1998

[SK98] J. Sairamesh, J. O. Kephart, "Price Dynamics of Vertically Differentiated Information Markets", Proc. of the 1st International Conference on Information and Computation Economics, Charleston, S.C., October 1998.

[SM02] H. Shimonishi, I. Maki, et al, "Dynamic fair bandwidth allocation for DiffServ classes", IEEE International Conference on Communications, Vol. 4, April-May 2002

[SP97] S. Shenker, C. Partridge, et al, "Specification of Guaranteed Quality of Service", RFC 2212, September 1997

[VK95] A. Vogel, B. Kerherve, et al, "Distributed multimedia and QOS: a survey", IEEE Multimedia, Vol. 2, Issue 2, 1995

[Wro97a] J. Wroclawski, "The Use of RSVP with IETF Integrated Services", RFC 2210, September 1997

[Wro97b] J. Wroclawski, "Specification of the Controlled-Load Network Element Service", RFC 2211, September 1997

[XN99] X. Xiao and L. Ni, "Internet QoS: A big picture", IEEE Network, March-April 1999

[YL03] M. Yang, E. Lu, et al, "Scheduling with Dynamic Bandwidth Allocation for DiffServ Classes", Proc. of the 12th International Conference on Computer Communications and Networks, Oct. 2003

Appendix A: Terminologies

Name	Definition
Bandwidth	Link transmission rate (bits/second).
Committed Information rate (CIR)	The average rate in bits per second at which the network agrees to accept data from a client. Data that is sent at a rate excess the CIR can be shaped or dropped if the network is congested.
DiffServ class	A service class that provides identical QoS for an aggregation of data flows belonging to this class.
DiffServ (DS) domain	"A contiguous set of nodes which operate with a common set of service provisioning policies and PHB definitions" [BBC98b]
DS behavior aggregate	An aggregation of data flow which have the DS field assigned and should be treated identically within a DS domain.
DS core router	A DS-compliant node that can perform DS core function to forward the packets.
DS edge router	"A DS-compliant node that connects one DS domain to a node either in another DS domain or in a domain that is not DS-capable" [BBC98b]
End-to-end delay (Delay)	The delay from source to destination including nodal processing delay, queuing delay, transmission delay, and propagation delay [KR03]
Flow	A flow is a sequence of packets with the same source and destination IP addresses, source and destination port numbers, and protocol ID.
Packet jitter (Jitter)	Variation of delays – the time from when a packet is generated at the source until it is received at the receiver can fluctuate from packet to packet [KR03].
Per-Hop Behavior (PHB)	"A description of the externally observable forwarding behavior applied at a DS-compliant node to a DS behavior aggregate." [BB98b]
RTP Session	"The association among a set of participants communicating with RTP. For each participant, the session is defined by the destination transport address (network address and port number). In the case of IP multicast, the destination transport address pair may be common for all participants." [SC03]
Service Level Agreement (SLA)	"SLA is a service contract between a customer and a service provider that specifies a forwarding service a customer should receive." [BB98b]
Throughput	The amount of data that can be sent from one location to another in a specific amount of time(Kbps, Mbps, or Gbps)
Traffic	An agreement specifying classifier rules, corresponding traffic

Conditioning Agreement (TCA)	profiles (a description of the temporal properties of a traffic stream such as rate and burst size), and conditioning rules (metering, marking, shaping, and policing).
User utility	A function of quality, price, and the weight reflecting the relative sensitivity to quality and price. It is used to control the quality and price tradeoff.

Appendix B: Abbreviations

Abbreviation	Full name
ACSA	Adaptive Class Switching Algorithm
AF	Assured Forwarding
ATM	Asynchronous Transfer Mode
BE	Best-effort
CIR	Committed Information Rate
CNAME	Canonical names
CR	DiffServ core router
CSA	Class Switching Algorithm
currClass	The class that our real-time multimedia data flow is running in
$currPktLoss_i$	The current packet loss experienced by the real-time multimedia flow running in class i
DCS	Dynamic Class Selection
DiffServ	Differentiated Services
DS	DiffServ
DSCP	DiffServ CodePoint
EF	Expedited Forwarding
ER	DiffServ edge router
IntServ	Integrated Services
ISP	Internet Service Provider
MAX_P	The price paid for the highest class AF4 which is the highest price
maxU	Maximum utility
MIN_Q	The minimum quality that a user is willing to tolerate and willing to pay for (MIN_Q = 1 – THD)
MRED	Multi-level RED

NUM_CLASS	Number of classes = 5
NS	Network Simulator
PHB	Per-Hop Behavior
p_i	The price paid for class i
PT	Packet Type
q_i	The QoS received by the receiver in class i
QoS	Quality of Services
RC	Reception Report Count
RED	Random Early Detection
RR	Receiver Report
RSVP	Resource ReSerVation Protocol
RTCP	Real Time Control Protocol
RTP	Real-time Transport Protocol
RTSP	Real-time Streaming Protocol
SDES	Source description
SLA	Service Level Agreement
SR	Sender Report
SSRC	Synchronization source identifier
switchClass	the class with the maxU
TCA	Traffic Conditioning Agreement
THD	Packet loss threshold
ToS	Type of Service
U	User utility
UDP	User Datagram Protocol
W	The weight of the quality

VITA AUCTORIS

NAME	Yang Feng
YEAR OF BIRTH	1970
PLACE OF BIRTH	Shenyang, P. R. China
EDUACTION	M.Sc., Computer Science University of Windsor Windsor, Ontario Canada 2005
	B.Sc., Computer Science University of Windsor Windsor, Ontario Canada 2002
	B.E., Mechanical Engineering Wuhan China 1992