Network Traffic Control and Bandwidth Management in Internet: A Differentiated Services Case Study

Suqiao Li

School of Computer Science
McGill University, Montreal

July, 1999

A thesis submitted to the
Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of
Master of Science

0-612-64392-1

Abstract

This Master's thesis deals with network traffic control and bandwidth management in Internet, and includes four parts. In part 1, we introduce network traffic, the basic principles of traffic control, control methods and components, which are used to support the Quality of Service (QoS) in different network environments. In part 2, we present network bandwidth management concepts and its methods. Bandwidth is finite and valuable, so we need to manage it efficiently. Bandwidth management can support traffic control to lighten the traffic load. Bandwidth management and traffic control are complementary, and together can assure a high QoS. In part 3, we address the QoS issues in Internet, since the Internet is increasingly important and popular. We focus on the Differentiated Services (DiffServ) in Internet, and implement the two-bit (Premium/Assured) based DiffServ by coordinated control. Coordinated control is the combination of traffic control, bandwidth control and queue control. Since the two-bit based DiffServ has two major drawbacks, in part 4, we propose some new methods and algorithms to improve them. These methods and algorithms include Multilevel Assured Service, Token-based Assured Service, Constraint Based Routing and load balancing.

Résumé

Ce mémoire de Maitrise comprend quatre parties. La première partie introduit les concepts du contrôle de trafic, ses méthodes et ses composants. La plupart d'entre eux sont utilisés pour supporter la Qualité de Service dans un environement réseau quelconque. La deuxième partie discute de la gestion de la bande passante du réseau et de ses méthodes. La bande
passante est finie et côuteuse, elle a donc besoin d'être gérée efficacement. La gestion de la bande passante peut supporter le contrôle du trafic pour alléger la charge du trafic. La gestion de la bande passante et le contrôle du trafic sont complémentaires et supportent tous la redisation de la Qualité de Service. A partir de la troisième partie, nous presentons la Qualité de Service au sein de l'Internet, vu la popularité grandissante de celui ci. On s'intérèsse particulièrement aux services différenciés sur l'Internet, pour les implémenter selon sur une architecture deux-bit, par un contrôle coordonné. Le contrôle coordonné est la combinaison du contrôle du trafic, du contrôle de la bande passante et du contrôle de la file d'attente. Il y a deux inconvénients majeurs aux services différenciés deux-bits. La quatrième partie présente de nouvelles méthodes et algorithms afin de l'améliorer. Ces méthodes compend le Service Assuré Multi-couches, le Service Assuré basé sur le jeton, le Routage basé sur la contrainte et l'équilibre de la charge.

Acknowledgements

I would like to express my thanks to my supervisor Dr. Petre Dini for his encouragement and sound judgement, and for giving me so many great opportunities to grow as a researcher under his thoughtful guidance. This thesis would not have been possible without his support. On a technical note, this thesis also benefited enormously from Andrei Negulescu, who gave me the most helpful guidance and suggestions. Special thanks are due to Prof. Newborn for his guidance during my stay at McGill University.

I also truly thank CRIM (Centre de recherche informatique de Montréal), which provides a so wonderful research environment for me.

I wish to thank the School of Computer Science for the graduate courses and the research environment. Thanks to Franca Cianci, Vicki Keirl, Lise Minogue, and Lucy St-James, for their great helps.

Many thanks also to Yihong Shangguan, Yuan Zhang, Huan Wang, Xiaobo Fan, Feng Xue, Mounina Bocoum, and Evelyna Evelguieva for their fruitful discussions within the group and for their kind encouragement.

Finally, my deepest thanks are due to Elaine X. Yu, for her endless support and amazing patience.

Table of Contents

List of Figures

List of Abbreviations

(in alphabetical order)

ATM:	Asynchronous Transfer Mode
BA:	Behavior Aggregate
BB:	Bandwidth Broker
BF:	Bellman-Ford's
DiffServ:	Differentiated Services
DS:	Differentiated Services
ECMP:	Equal-Cost Multi-Path
FIFO:	First In First Out
GDS:	Generalized Differentiated Services
IETF:	Internet Engineering Task Force
IntServ:	Integrated Services
IP:	Internet Protocol
ISP:	Internet Service Provider
LANs:	Local Area Networks
MANs:	Metropolitan Area Networks
MF:	Multi-Field
MPLS:	Multi-Protocol Label Switching
OSPF:	Open Shortest Path First
QoS:	Quality of Service
PHB :	Per-Hop-Behavior
RED:	Random Early Detection
RIO:	Random Early Detection with In and Out
RIP:	Routing Information Protocol
RSVP:	ReSerVation Protocol
SDS:	Specialized Differentiated Services

SLA:	Service Level Agreement
TDM:	Time Division Multiplexing
VR:	Virtual Reality
WAN:	Wide Area Networks
WFQ:	Weighted Fair Queuing
WRR:	Weighted Round Robin

Chapter 1 Introduction

Nowadays telecom networks are increasingly complex because they make use of many protocol and network architectures, such as Ethernet, Token Ring, FDDI, SONET, ATM, Internet, and so on [1]. Different networks provide different services for users. In order to get good Quality of Service (QoS) from the network, we need to control the network traffic and to manage the available bandwidth, because ***"no control, no service; good control, good service."***

We will discuss the network traffic control and some control methods in Chapter 2, bandwidth management and some methods in Chapter 3. All these methods can be used to support QoS. QoS is our goal for building and managing the networks [2]. Generally, QoS has several levels: best-effort services, differentiated services, and guaranteed services. QoS has a tight relationship with traffic control and bandwidth management.

We present an adaptive bandwidth management architecture for the Internet in Chapter 3. The concepts behind this architecture are used in Chapter 5 to improve the two-bit based Differentiated Services.

Since the Internet is so important and popular, many new services need to be implemented in the Internet, such as multimedia communication, videoconferencing, Internet telephony, etc. But, the current Internet cannot satisfy these requirements because of its best-effort basis, where all packets are treated equally. The Internet Engineering Task Force (IETF) has defined a new service for the Internet--Differentiated Services (DiffServ)-- but this service has not yet been standardized. There are some architectures supporting the DiffServ in the Internet, like the two-bit (Premium/Assured) architecture [3]. The implementation of two-bit architecture is addressed in Chapter 4. DiffServ is implemented by coordinated control, which is the combination of traffic control, bandwidth control and queue control. The relationship between DiffServ and coordinated control is described in Chapter 4.

The two-bit architecture has some drawbacks. In this thesis, we focus on the two most important ones. One drawback is the lack of scalability of service quality, and the other one is the lack of high utilization of network resources. We build new models and propose new algorithms to solve these problems in Chapter 5. The algorithms have been implemented in Java. The Java simulation program may be found in the Appendix. The conclusion and future work are described in Chapter 6.

1.1 Network Traffic Control

Network traffic control is a set of policies and mechanisms that allows a network to efficiently satisfy a diverse range of service requests [4]. There are two fundamental aspects of traffic control, diversity in user requirements and efficiency in satisfying them. Traffic control includes a rich set of mechanisms, such as traffic shaping, scheduling, monitoring, policing, signaling, pricing, admission control, congestion control and flow control, etc. In another way, traffic control refers to a set of actions taken by the network to avoid congested conditions, which shapes the behavior of data flows at the entry, and at specific points within the system [1].

Traffic control allows a network to give the most utility with the available resources. Traffic control consists of a collection of specification techniques and mechanisms to:

- Specify the expected characteristics and requirements of a data stream;
- Shape data streams at the edges and selected points within the queuing network;
- Police data streams and take corrective actions when traffic deviates from its specification.

There are three general laws for traffic control:

1) The network should try to match its menu of service qualities to user requirements. Service menus that are more closely aligned with user requirements are more efficient.
2) Building a single network that provides heterogeneous QoS is better than building separate networks for different levels of QoS.

3) For typical utility functions, if network utilization remains the same, the sum of user utility functions increases more than linearly with an increase in network capacity.

To effectively control traffic, a network provider must know not only the requirements of individual applications and organizations, but also their typical behavior. A traffic model summarizes the expected behavior of an application or an aggregate of applications.

Networks that provide heterogeneous QoS are likely to cost less than networks that provide a single QoS. Traffic classes represent the shared requirements of a set of widely used applications, which also represent the types of service provided by the network. We partition applications into two fundamental classes: "guaranteed" and "best-effort".

The "guaranteed" applications include videoconferencing, telephony, remote sensing, video-on-demand, interactive multi-player games, etc. With these applications, users derive utility from the network only if the network limits the delay and provides a minimum amount of bandwidth. The utility function for a "guaranteed" application penalizes traffic that does not meet its service requirement, which is typically described by three parameters: bandwidth, delay, and loss.

The "best-effort" applications are willing to adapt to whatever QoS is available. The utility function for a best-effort application does not degrade significantly with a drop in service quality. Unlike a guaranteed application, a best-effort application derives utility from the network even if its packets suffer long delays, or it receives only a small bandwidth allocation from network.

To conclude, the above are the general concepts of network traffic control. We will describe the traffic control principles and control methods in Chapter 2.

1.2 Bandwidth and Bandwidth Management

Bandwidth is an important system resource [5]. In digital systems, bandwidth is data speed in bits per second (bps). Thus, a modem that works at 57,600 bps has twice the bandwidth of a modem that works at 28,800 bps. In analog systems, bandwidth is defined in terms of the difference between the highest-frequency signal component and the lowest-frequency signal component. Frequency is measured in cycles per second (Hertz).

Generally speaking, bandwidth is directly proportional to the amount of data transmitted or received per unit time. In a qualitative sense, bandwidth is proportional to the complexity of the data for a given level of system performance. For example, it takes more bandwidth to download a photograph in one second than it takes to download a page of text in one second. Large sound files, computer programs, and animated videos require still more bandwidth for acceptable system performance. Virtual reality (VR) and full-length three-dimensional audio/visual presentations require the most bandwidth of all.

Until now, and at the present time bandwidth has been and is a finite and important system resource, even for local networks. Cheap and abundant bandwidth may be available in the future [6], but in the present we have to manage the bandwidth, to save it and to use it economically and efficiently, therefore we need bandwidth management.

Bandwidth management is another important way to ensure the QoS. We consider that there is a tight relationship between bandwidth and QoS. It is easier to get good QoS if the bandwidth is large. Bandwidth management may support traffic control. When the bandwidth is abundant, traffic control will be easier, because there is no need to use complicated traffic control methods.

In short, the wider the bandwidth, the easier the traffic control and the better the QoS. We will propose some methods for bandwidth management and present an active bandwidth management architecture for Internet in Chapter 3.

1.3 Quality of Service (QoS)

QoS refers to the ability of a network to provide better service to selected network traffic over various underlying technologies including Frame Relay, Asynchronous Transfer

Mode (ATM), Ethernet and 802.1 networks, SONET, and IP-routed networks [7]. In particular, QoS features provide better and more predictable network service by:

- Supporting dedicated bandwidth;
- Improving loss characteristics;
- Avoiding and managing network congestion;
- Shaping network traffic;
- Setting traffic priorities across the network.

Traffic control and bandwidth management are two major ways to achieve the QoS. QoS is core for traffic and bandwidth control.

1.3.1 QoS Architecture

We configure QoS features throughout a network to provide for end-to-end QoS delivery [7]. The following components are necessary to deliver QoS across a heterogeneous network: (1) QoS within a single network element, which includes queuing, scheduling, and traffic shaping features. (2) QoS signaling techniques for coordinating QoS between network elements. (3) QoS policing and management functions to control and administer end-to-end traffic across a network.

Not all QoS techniques are appropriate for all network routers. Because edge routers and backbone routers in a network do not necessarily perform the same operations, the QoS tasks they perform might differ as well.

Generally, edge routers perform the following QoS functions:

- Packet classification;
- Admission control;
- Configuration management.

Generally, backbone routers perform the following QoS functions:

- Congestion management;
- Congestion avoidance.

1.3.2 End-to-End QoS Models

End-to-end QoS is the ability of a network to deliver service required by specific network traffic from one end of the network to another. There are three main types of service models: best effort, integrated, and differentiated services [7].

A) Best-effort Service

Best-effort is a single service model in which an application sends data whenever it must, in any quantity, and without requesting permission or first informing the network. For best-effort service, the network delivers data if it can, without any assurance of reliability, delay bounds, or throughput. A large amount of Internet traffic nowadays is best-effort based.

B) Integrated Service

Integrated service is a multiple service model that can accommodate multiple QoS requirements. In this model the application requests a specific kind of service from the network before sending data. The request is made by explicit signaling; the application informs the network of its traffic profile and requests a particular kind of service that can encompass its bandwidth and delay requirements. The application is expected to send data only after it gets a confirmation from the network. It is also expected to send data that lies within its described traffic profile. The network performs admission control, based on information from the application and available network resources. It also commits to meet the QoS requirements of the application as long as the traffic remains within the profile specifications. The network fulfills its commitment by maintaining per-flow state and then performing packet classification, policing, and intelligent queuing based on that state.

C) Differentiated Service

Differentiated service is a multiple service model that can satisfy different QoS requirements. However, unlike the integrated service model, an application using differentiated service does not explicitly signal the router before sending data. For differentiated service, the network tries to deliver a particular kind of service based on the QoS specified by each packet. This specification can occur in different ways. For example, using the IP Precedence bit settings in IP packets or source and destination addresses. The network uses the QoS specification to classify, shape, and police traffic, and to perform intelligent queuing. The differentiated service model is used for several mission-critical applications and for providing end-to-end QoS. Typically, this service model is appropriate for aggregate flows because it performs a relatively coarse level of traffic classification.

We focus on differentiated service in the Internet in Chapter 4 and 5, because we consider that differentiated service is easier to implement than integrated service. We believe that differentiated service is the first step of a wide-scale QoS in the Internet.

1.3.3 QoS Broker

The QoS Broker provides end-to-end guarantees, balances resources among applications, the network and operating systems at end-points, and between end-points and the network. It manages resources at the end-points, coordinating resource management across layer boundaries. As an intermediary, it hides implementation details from applications and per-layer resource managers.

A QoS Broker uses translation, admission and negotiation services to configure the system to application needs. Configuration is achieved via QoS negotiation resulting in one or more connections through the communications system. The negotiation involves all components of the communication system needed for the setup [8].

The roles of the QoS Broker may include: (1) managing resources needed for tasks in the application and transport subsystems at the end-points, (2) negotiating with network resource management, and (3) negotiating with remote QoS Brokers.

An important property of the QoS Broker is its role as an active intermediary which insulates cooperating entities from the operational details of other entities. The QoS Broker manages communications among different entities to create the desired system configuration.

1.4 The Relationship Between QoS, Traffic Control and Bandwidth Management

Traffic control and bandwidth management are complementary in supporting a high QoS. The better the traffic control and bandwidth management, the better the QoS. The relationship of QoS, traffic control and bandwidth management is shown in Figure 1.1. Traffic control and bandwidth management combine together to ensure a satisfactory level QoS for the customer.

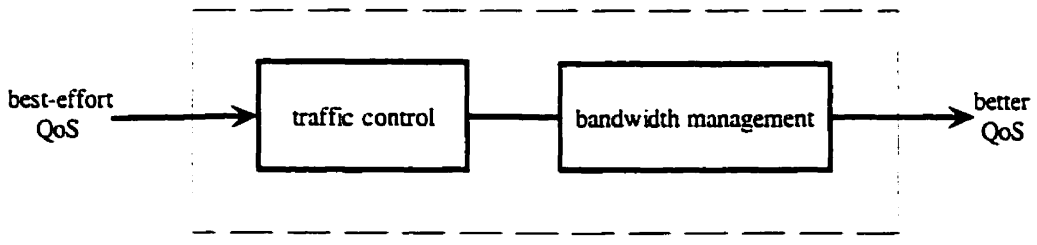


Figure 1.1 The relationship between QoS, traffic control and bandwidth management

In the following chapters, we will describe the principle of traffic control and its methods, bandwidth management and its methods, differentiated services in Internet, and improving two-bit differentiated service architecture, respectively.

Chapter 2 The Principle of Traffic Control and its Methods

In this chapter, we discuss the principle of traffic control, some control methods and control components. All of these concepts are used in Chapter 4.

2.1 Principle of Traffic Control

Based on control theory, there are two kinds of control mechanisms: (1) open loop, (2) closed loop.

2.1.1 Open-loop Control

Open loop control is based on good design. When designing the control system, we have to make sure that all kinds of problems will not occur in the first place. Once the system is up and running, corrections cannot be made. An open-loop control system does not compare the actual result with the desired result to determine the control action. Instead, a calibrated setting is used to obtain the desired result. The primary advantage of open-loop control is that it is less expensive than closed-loop control, as there is no need to measure the actual result. In addition, the controller is much simpler because corrective action based on the error is not required. The disadvantage of open-loop control is that errors caused by unexpected disturbances are not corrected [9].

2.1.2 Closed-loop Control

In contrast, closed-loop control is based on the concept of a feedback loop. Feedback is the action of measuring the difference between the actual result and desired result, and using the difference to drive the actual result toward the desired result. The term feedback comes from the direction in which the measured value signal travels in the block diagram. The signal begins at the output of the controlled system and ends at the input to the controller. The output of the controller is the input to the controlled system. Thus the measured value signal is fed back from the output of the controlled system to the input.

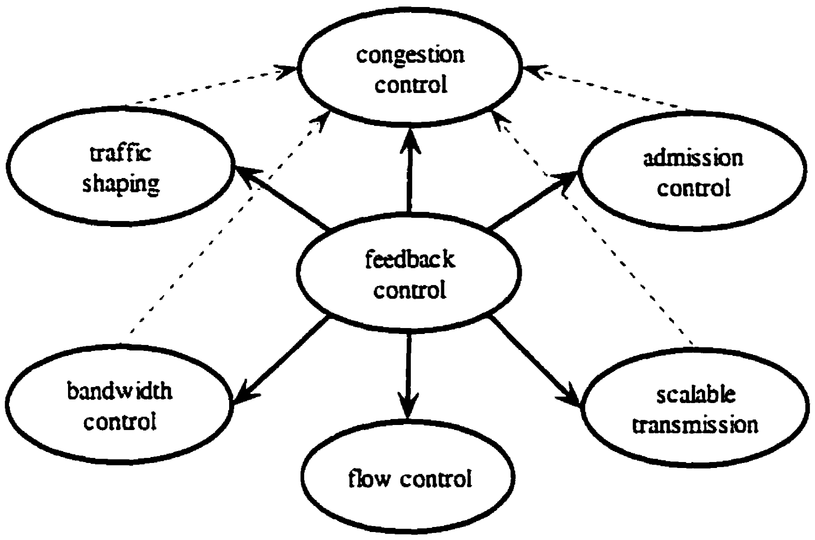


Figure 2.1 The relationship between feedback control and various components of traffic Control [1]

Both open loop and closed loop control methods must keep the whole system stable. Some situations, such as congestion, can make the system unstable, so the system needs to control congestion [1].

Feedback is the core of closed-loop control. The relationship between feedback control and various components of traffic control is shown in Figure 2.1. Traffic control and congestion control must be distinguished, as they are two different concepts. Congestion control is just one aspect of traffic control.

2.2 Congestion Control

Congestion control is an important problem for network traffic control. Many articles have addressed the issue of congestion control in almost every kind of network, such as congestion control in ATM networks, congestion control in the Internet, and so on [10,11]. Congestion control refers to the set of actions taken by the network to minimize the intensity, spread, and the duration of congestion. Congestion can be caused by unpredictable statistical fluctuations of traffic flows and fault conditions within the network. Congestion is a phenomenon where the amount of traffic injected into the network exceeds the capacity of the network [9].

Congestion control includes two parts:

1. The network must be able to signal the transport endpoints that congestion is occurring or about to occur.
2. The endpoints must have a policy that decrease utilization if this signal is received and increases utilization if the signal is not received.

Congestion control can be achieved using either open-loop or closed-loop mechanisms. The open-loop scheme is based on designing and configuring the system carefully to avoid the occurrence of congestion. The closed-loop scheme is based on feedback. We rather prefer to use the closed-loop scheme.

2.2.1 Closed-loop Congestion Control

For the closed-loop feedback control system, there are three steps to approaching congestion control:

1. Monitoring the system to detect when and where congestion occurs.
2. Passing this information to places where action can be taken.
3. Adjusting system operation to correct the problem.

Congestion control schemes try to protect the shared network resources from saturation by dynamically adjusting the traffic of the network. Congestion control schemes can be classified into two categories: reactive control and preventive control. With reactive control, sources adjust their traffic flows based on feedback information received from the network about the presence of congestion. With preventive control, sources must reserve network resources in advance, before they can access the network, and are required to remain within their allocated resources [12].

Various metrics can be used to monitor the congestion. Chief among these are the percentage of all packets discarded for lack of buffer space, the average queue lengths, the number of packets that time out and are retransmitted, the average packet delay, and the standard deviation of packet delay. In all cases, rising numbers indicate growing congestion. The second step in the feedback loop is to transfer the information about the congestion from the point where it is detected to the point where something can be done about it. The obvious way is for the router detecting the congestion to send a packet to the traffic source, announcing the problem. Of course, these extra packets increase the load at precisely the moment that more load is not needed as congestion is happening.

2.2.2 Congestion Detection and RED

Congestion grows exponentially, and must be detected as early as possible [14]. We describe some methods to detect congestion below.

- The most common method in use is to notice that the output buffers at a switch are full, and there is no space for incoming packets. If the switch wishes to avoid packet loss, congestion avoidance steps can be taken when some fraction of the buffers are full. A time average of buffer occupancy can help smooth transient spikes in queue occupancy.
- A switch may monitor output line usage. It has been founded that congestion occurs when trunk usage goes over a threshold and so this metric can be used as a signal of impending congestion.
- A source may monitor round-trip delays. An increase in these delays signals an increase in queue sizes, and possible congestion.
- A source may probe the network's state using some probing scheme.

- A source can keep a timer that sets off an alarm when a packet is not acknowledged in time. When the alarm goes off, congestion is suspected.

Random Early Detection (RED) is a useful method for congestion detection. RED improves on early random drop in three ways. First, pack ets are dropped based on an exponential average of the queue length, rather than the instantaneous queue length. This allows small bursts to pass through unharmed, dropping packets only during sustained overloads. Second, the packet drop probability is a linear function of the average queue length. As the mean queue length increases, the probability of packet loss increases. This prevents a severe reaction to mild overload (as with early random drop). Finally, RED switches can not only drop packets, but mark offending packets. With suitably modified endpoints, RED switches allow congestion avoidance similar to the DECbit scheme [25].

One method for gateways to notify the source of congestion is to drop packets. This is done automatically when the queue is full. The default algorithm is, when the queue is full, to drop the any new packets. This is called tail-drop. Another algorithm is when the queue is full and a new packet arrives, one packet is randomly chosen from the queue to be dropped. This is called random-drop. The drawback to tail-drop and random-drop gateways is that it drops packets from many connections and causes them to decrease their windows at the same time resulting in a loss of throughput.

Early-random-drop gateways are a slight improvement over tail-drop and random-drop in that they drop incoming packets with a fixed probability whenever the queue size exceeds a certain threshold.

- **Algorithm of RED**

In RED method, once the average queue is above a certain threshold the packets are dropped with a certain probability related to the queue size. To calculate the average queue size the algorithm uses an exponentially weighted moving average:

$$avg = (1-wq)avg + wq*Queue\_Size$$

The probability to drop a packet, pb, varies linearly from 0 to maxp as the average queue length varies from the minimum threshold, minth, to the maximum threshold, maxth. The chance that a packet is dropped is also related to the size of the packet. The probability to drop an individual packet, pa, increases as the number of packets since the last dropped packet, count, increases:

$$pb = maxb(avg\text{-}minth)/(maxth\text{-}minth)$$
$$pb = pb*Packet\_Size/Max\_Packet\_Size$$
$$pa = pb/(1\text{-}count*pb)$$

In this algorithm as the congestion increases, more packets are dropped. Larger packets are more likely to be dropped than smaller packets that use less resources [25].

- **Advantages of RED**

There are several advantages of RED:

1) Absorbs bursts better;
2) Avoids synchronization;
3) Signals end systems earlier.

2.2.3 Congestion Communication

Communication of congestion information from the congested switch to a source can be implicit or explicit. When communication is explicit, the switch sends information in packet headers or in control packets such as source quench packets, choke packets, state-exchange packets, rate-control messages, or throttle packets to the source [15].

Implicit communication occurs when a source uses probe values, retransmission timers, throughput monitoring, or delay monitoring to indicate the occurrence of congestion. Explicit communication imposes an extra burden on the network, since the network needs to transmit more packets than usual, and this may lead to a loss in efficiency. On the other hand, with implicit communication, a source may not be able to distinguish between congestion and other performance problem, such as a hardware problem. Thus, the communication channel is quite noisy, and a cause of potential instability.

2.2.4 Congestion Pricing

The first economic principle is that there is only a marginal cost to carrying a packet when the network is congested. When congestion happens, the cost of carrying a packet from user A is the increased delay seen by user B. The traffic of user B, of course, caused delay for A. But if A somehow were given higher priority, so that B saw most of the delay, A would be receiving better service, and B paying a higher price, in terms of increased delay and dissatisfaction. According to economic principles, A should receive better service only if he is willing to pay enough to exceed the "cost" to B of his increased delay. This can be achieved in the marketplace by the setting of suitable prices. In principle, one can determine the pricing for access dynamically by allowing A and B to bid for service, although this has many practical problems. When the network is under-loaded, however, the packets from A and from B do not interfere with each other. The marginal or incremental cost to the service provider of carrying the packets is zero. In a circumstance where prices follow intrinsic costs, the usage-based component of the charge to the user should be zero. This approach is called congestion pricing [13].

2.2.5 Congestion Control Algorithms

The closed loop algorithms are divided into two subcategories (1) explicit feedback, (2) implicit feedback.

- Explicit feedback: packets are sent back from the point of congestion to warn the source.
- Implicit feedback: the source deduces the existence of congestion by making local observations, such as the time needed for acknowledgements to come back.

The presence of congestion means that the load is greater than the resources can handle. Two solutions can be used: increase the resources or decrease the load. Splitting traffic over multiple routes instead of always using the best one may also effectively increase the bandwidth. Spare routers that are normally used only as backups can be put on-line to give more capacity when serious congestion appears. However, sometimes it is not possible to increase the capacity, or it has already been increased to the limit. The only way then to beat back the congestion is to decrease the load. Several ways exist to reduce the load, including denying service to some users, degrading service to some or all users, and having users schedule their demands in a more predictable way.

2.2.6 Decongestion

An overloaded switch can signal impending congestion to the sources, and, at worst, can drop packets. If buffer usage is a congestion metric, switches drop packets or throttle sources when a source exceeds its share of buffers. This share is determined by the buffer allocation strategy, and the rate at which the buffers are emptied depends on the service discipline. Thus, the buffer allocation strategy and the service discipline jointly determine which sources are affected [16].

2.2.7 Flow Control over Congestion Control Scheme

A number of congestion control schemes have been proposed that operate at the sources. These schemes use the loss of a packet to reduce the source sending-rate in some way. The two main types of schemes are choke schemes and rate-control schemes. In a choke scheme, a source shuts down when it detects congestion. After some time, the source is allowed to start again. Choking is not efficient, since the reaction of the sources is too abrupt. In a rate-control scheme, when a source detects congestion it reduces the rate at which it sends out packets, either using a window adjustment scheme or a rate adjustment scheme. The advantage of rate control schemes over choke schemes is that rate control allows a gradual transition between sending no packets at all to sending out packets full blast [17].

2.3 Flow Control

Flow control refers to the set of techniques that enable a data source to match its transmission rate to the currently available service rate at a receiver and in the network. Besides this primary goal, a flow control mechanism should meet several other, sometimes mutually contradictory objectives. It should be easily implemented so that the least possible network resources are used, and to work effectively even when used by many sources. If possible, each member of the entire set of flow-controlled sources sharing a scarce resource should restrict its usage to its fair share. Finally, the set of sources should be stable, so when the number of sources is fixed, the transmission rate of each source settles down to an equilibrium value. Stability also implies that, if a new source becomes active, existing active sources adjust their transmission rates, and, after a brief transition period, the system settles down to a new equilibrium. We can implement flow control at the application, transport, network, or data link layer of a protocol stack. The choice of layer depends on the situation at hand. The most common design is to place end-to-end flow control at the transport layer, and hop-by-hop flow control in the data link layer [18].

Flow control is often confused with congestion control. Congestion refers to a sustained overload of intermediate network elements. Thus, flow control is one mechanism for congestion control. We can divide flow control techniques into three broad categories: open loop, closed loop, and hybrid.

2.3.1 Open-loop Flow Control

In open-loop flow control, a source has to describe its entire future behavior with a handful of parameters, because the network's admission control algorithm uses these parameters to decide whether to admit the source or not. Open-loop flow control works best when a source can describe its traffic well with a small number of parameters, and when it needs to obtain QoS guarantees from the network. If either of these conditions fails to apply, the source is better off with closed-loop or hybrid flow control [15].

2.3.2 Closed-loop Flow Control

In closed-loop flow control, we assume that network elements do not reserve sufficient resources for the connection, either because they do not support resource reservation, or because they overbook resources to get additional statistical multiplexing. Some protocols can be used for close-loop flow control, as follows:

- **On-off flow control**

In on-off flow control, the receiver sends the transmitter an On signal when it can receive data, and an Off signal when it can accept no more data. The transmitter sends as fast as it can when it is in the On state, and is idle when it is in the Off state. On-off control is effective when the delay between the receiver and the sender is small. It works poorly when the propagation delay between the sender and receiver is large, because the receiver needs to buffer all the data that arrive before the Off signal takes effect [1].

- **Stop-and-wait**

In the stop-and-wait protocol, one of the earliest attempts at flow control, a source sends a single packet and waits for an acknowledgment before sending the next packet. If it received no acknowledgment for some time, it times out and retransmits the packet. Stop-and-wait simultaneously provides error control and flow control. It provides error control because if a packet is lost, the source repeatedly retransmits it until the receiver acknowledges it. It provides flow control because the sender waits for an acknowledgment before sending a packet. Thus, stop-and-wait forces the sender to slow down to a rate slower than can be supported at the receiver [1].

- **DECbit flow control**

The key idea behind the DECbit scheme is that every packet header carries a bit that can be set by an intermediate network element that is experiencing congestion. The receiver copies the bit from a data packet to its acknowledgment, and sent the acknowledgment back to the source. The source modifies its transmission-window size based on the series of bits it receives in the acknowledgment header as follows: The source increases its windows until it starts building queues at the bottleneck server, causing the server to set bits on the source's packets. When this happens, the source reduces its window size, and bits are no longer set [20].

In the DECbit scheme, each network element monitors packet arrival from each source to compute its bandwidth demand and the mean aggregate queue length. The DECbit scheme has several useful properties. It requires only one additional bit in the packet header and does not require per-connection queuing at servers. Endpoints can implement the scheme in software, without additional hardware support.

- **TCP flow control**

The flow-control scheme in TCP is similar to the DECbit scheme, but differs in one important detail. Instead of receiving explicit congestion information from network elements, a source dynamically adjusts its flow control window in response to implict signals of network overload.

2.3.3 Hybrid Flow Control

In open-loop flow control, a source reserves capacity according to its expected traffic, whereas in closed-loop flow control, the source must adapt to changing network conditions. In hybrid control, a source reserves some minimum capacity, but may obtain more if other sources are inactive. Hybrid control schemes not only inherit the problems of open-loop and closed-loop control, but also introduce some new ones. Source descriptors in hybrid control can be less stringent than in open-loop control, because its descriptor does not limit a source. Hybrid controlled sources must obey all appropriate closed-loop control mechanisms. Hybrid control has a strong advantage: a guaranteed minimum resource allocation to an admitted packet, even when the network is overloaded. Thus, a hybrid-controlled source, once admitted, knows that even in the worst case, it has some minimum bandwidth guaranteed to it, and that in the average case, it will obtain substantially more bandwidth [15].

To sum up, despite having all the problems of open-loop and close-loop flow control, hybrid control has the advantage of being able to guarantee minimum service rate to admitted calls even in the worst case.

2.4 Traffic Descriptors

A traffic descriptor is a set of parameters that describes the behavior of a data source. Typically, it is a behavior envelope, describing the worst possible behavior of a source, rather than its exact behavior. A descriptor plays three roles besides describing source traffic. First, it forms the basis of a traffic control between the source and the network: the source agrees not to violate the descriptor, and in turn, the network promises a particular QoS. Second, the descriptor is the input to a regulator, a device through which a source can pass data before it enters the network. To ensure that the source never violates its traffic descriptor, a regulator delays traffic in a buffer when the source rate is higher than expected. Third, the descriptor is also the input to a policer, a device supplied by the network operator that ensures that the source meets its portion of the contract. A policer delays or drops source traffic that violates the descriptor. The regulator and policer are identical in the way they identify descriptor violations: the difference is that a regulator typically delays excess traffic, while a policer typically drops it [21].

A practical traffic descriptor must have these important properties:

- Representativity: The descriptor must adequately represent the long-term behavior of the traffic, so that the network does not reserve too little or too much.
- Verifiability: The network must be able to verify quickly, cheaply, and preferably in hardware that a source is obeying its promised traffic specification.
- Preservability: The network may inadvertently modify source traffic behavior as it travels along its path. Thus, the amount of resources allocated to a channel may change the path. The network must be able either to preserve the traffic characteristics along the path, or to calculate the resource requirements of the modified traffic stream.
- Usability: Sources should be able to describe their traffic easily, and network elements should be able to perform admission control with the descriptor easily.

Coming up with good traffic descriptors is difficult because of these conflicting requirements. We choose the source's peak rate as the descriptor. It is usable, preservable, and verifiable, but not representative, because resource reservation at the peak rate is wasteful if a source rarely generates data at this rate. There are two common descriptors: peak rate and average rate.

2.4.1 Peak Rate

The peak rate is the highest rate at which a source can ever generate data during a packet. A trivial limit on the peak rate of a connection is the speed of the source's access link, because this is the instantaneous peak rate of the source during actual packet transmission. For networks with fixed-size packets, the peak rate is the inverse of the closest spacing between the starting times of consecutive packets. For variable-sized packets, we must specify the peak rate along with a time window over which we measure this peak rate. Then, the peak rate limits the total number of packets generated over all windows of the specified size. A peak-rate regulator consists of a buffer and a timer. For the moment, assume a fixed size packet network. When the first packet arrives at the buffer, the regulator forwards the packet and sets a timer for the earliest time it can send the next packet without violating the peak-rate bound, that is, the smallest inter-arrival time. It delays subsequently arriving packets in a data buffer until the timer expires. If the timer expires before the next packet arrives, it restarts the timer on packet arrival, and the incoming packet is forwarded without delay [22].

The peak-rate descriptor is easy to compute and police. Peak-rate descriptors are useful only if the traffic sources are very smooth, or if a simple design is more important than efficiency.

2.4.2 Average Rate

The key problem with the peak rate is that it is subject to outliers. The motivation behind average-rate descriptors is that averaging the transmission rate over a period of time reduces the effect of outliers. Two types of average-rate mechanisms have been proposed. Both mechanisms use two parameters, T and A, defined as follows:

T = time window over which the rate is measured;

A = the number of bits that can be sent in a window of time T.

In the jumping-window descriptor, a source claims that over consecutive windows of length *T* seconds, no more than *A* bits of data will be transmitted. The term "jumping window" refers to the fact that a new time interval starts immediately after the end of the earlier one. The jumping-window descriptor is sensitive to the choice of the starting time of the first window.

In the moving-window scheme, the time window moves continuously, so that the source claims that over all windows of length t seconds, no more than *A* bits of data will be injected into the network. The moving-window scheme removes the dependency on the starting time of the first window. It also enforces a tighter bound on spikes in the input traffic. An average-rate regulator is identical to a variable-packet-size peak-rate regulator, because both restrict the maximum amount of information that can be transmitted in a given interval of time. For a jumping-window descriptor, at time 0, a counter is initialized to 0 and is incremented by the packet size of each departing packet. Every *T* seconds, the counter is reset to 0.When a packet arrives, the regulator computes whether sending the packet would result in too much data being sent in the current window. This test reduces to testing whether the sum of the current counter value and the current packet size is larger or smaller than *A*. Depending on the result, the regulator either forwards the packet immediately or buffers it until the next time window [22].

2.5 Traffic Shaping

Traffic shaping can be done either at the end systems, or in the network by the switch hardware. Traffic shaping at the end systems can be implemented by the server using a Leaky Bucket (single or dual) shaper consisting of a buffer and a rate controller. The main issues are the rate control mechanism, shaper delay and delay variation, and the shaper buffer size at the server. The rate controller determines the outgoing data rate which should be consistent with the bandwidth available from the network. An easy-to-implement set of traffic descriptors is therefore a key factor in obtaining good performance from the shaper. Close-loop feedback rate control which utilizes feedback obtained from the network can be used to control the traffic rate [23].The shaper needs a large buffer for accumulating the incoming bursty stream. However, if the outgoing rate of the shaper is low, a large shaper buffer may result in long delay variation. Therefore, there exists a trade-off between the buffer size, shaper delay, and outgoing rate of the shaper.

Traffic shaping limits the data transmission rate. We can limit the data transfer to a specific configured rate, or a derived rate based on the level of congestion. As mentioned, the rate of transfer depends on these three components that constitute the token bucket: burst size, mean rate, measurement interval. The mean rate is equal to the burst size divided by the interval. When traffic shaping is enabled, the bit rate of the interface will not exceed the mean rate over any integral multiple of the interval. In other words, during every interval, a maximum burst size can be transmitted. Within the interval, however, the bit rate may be faster than the mean rate at any given time [24].

Traffic shaping smoothes traffic by storing traffic above the configured rate in a queue. When a packet arrives at the interface for transmission, the following happens:

- If the queue is empty, the arriving packet is processed by the traffic shaper. If possible, the traffic shaper sends the packet. Otherwise, the packet is placed in the queue.
- If the queue is not empty, the packet is placed in the queue.

When there are packets in the queue, the traffic shaper removes the number of packets it can transmit from the queue every time interval [23].

2.6 Traffic Scheduling

Scheduling disciplines such as weighted fair queuing and rate-controlled static priority scheduling allow individual connections to obtain guarantees on bandwidth, delay, and delay jitter. Thus, packets from guaranteed-service sources should be scheduled according to one of these disciplines. These sources should reserve enough resources to meet their performance requirements.

Scheduling should meet not only individual, but also organizational performance requirements. Note that a conflict between individual and organizational performance requirements is possible, in that a packet might need to be given a low delay to meet its delay bound, but the connection on which the packet arrived might have already used its bandwidth quota. If the scheduler delays the packet, the organizational performance requirement is met, but the individual performance requirement is not. If the scheduler sends the packet before its deadline, the opposite holds is true.

2.7 Traffic Policing

Since the network must protect guaranteed-service clients from malicious users, it needs to monitor the traffic from each source to ensure that it satisfies its traffic specification. Such an access control function at the network's edge is called policing. The input to the policer comes from the source, and the output goes to the network. The function of the policer is to ensure that the traffic it outputs to the network satisfies the traffic constraint function. To achieve this, the policer may need to buffer or drop packets when the input stream exceeds the limit. If the input stream to the source policer satisfies the traffic constraint function, no buffering or delay is incurred in the policer [19].

2.7.1 Leaky Bucket Policing and Algorithm

Effective policing of traffic can prevent congestion from occurring and therefore a policing function that controls traffic to the reliability level necessary is a crucial requirement. One such policing requirement, known as the Leaky Bucket policing function, has the potential to meet this critical demand.

Each host is connected to the network by an interface containing a leaky bucket, which is a finite internal queue. When a packet arrives, if there is room on the queue it is appended to the queue; otherwise, it is discarded [27]. Leaky Bucket Algorithm enforces a rigid output pattern at the average rate, no matter how bursty the traffic is. For many situations, it is better to allow the output to speed up somewhat when large bursts arrive, such as in the Token Bucket Algorithm.

2.7.2 Token Bucket Algorithm

The leaky bucket holds tokens, generated by a clock at the rate of one token every N seconds. For a packet to be transmitted, it must capture and destroy one token. The token bucket algorithm provides a different kind of traffic shaping than the leaky bucket algorithm, which does not allow idle hosts to save permission to send large bursts later. The token bucket algorithm does allow saving, up to the maximum size of the bucket. This property means that bursts of up to the maximum packets can be sent at once, allowing tolerance for bursts in the output stream and giving faster response to sudden bursts of input. Another difference between the two algorithms is that the token bucket algorithm throws away tokens when the bucket fills up but never discards packets [28].

The leaky bucket and token bucket algorithms can be used to design the traffic shaper. A shaper based on token bucket algorithm is shown in Figure 2.2.

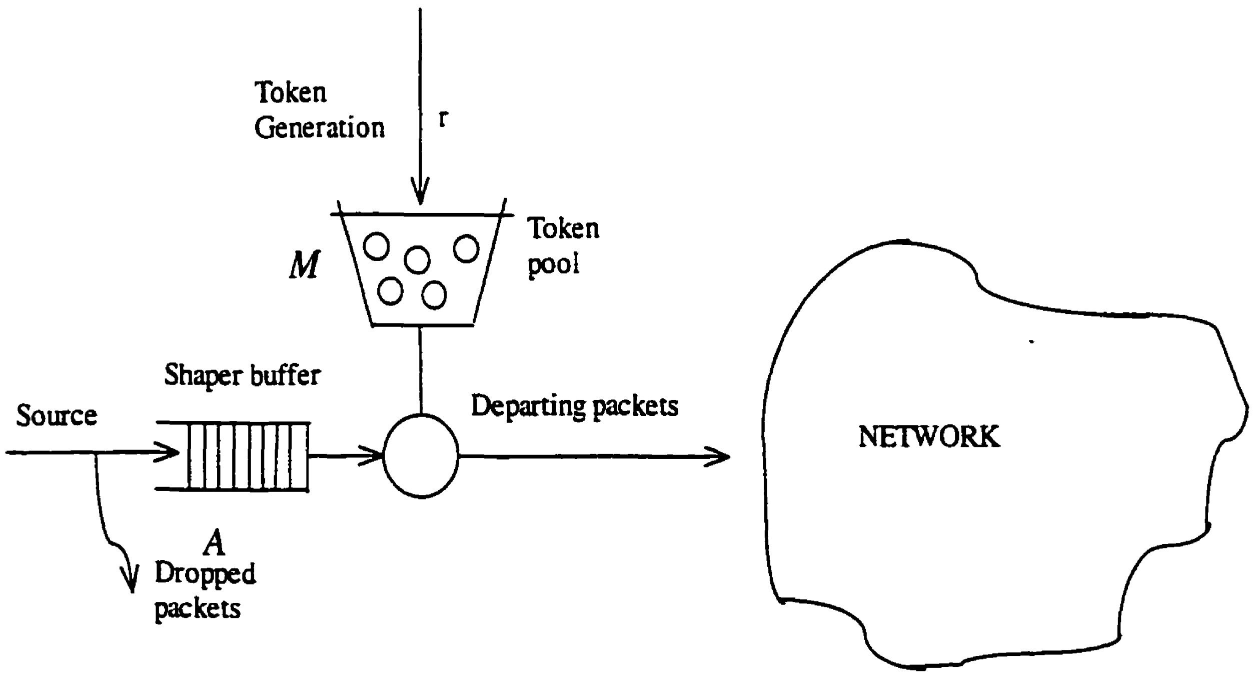


Figure 2.2 A shaper based on Token Bucket Algorithm

2.8 Traffic Signaling

Signaling is the process by which an endpoint requests the network to set up, tear down, or renegotiate a request. Two distinct mechanisms are involved in signaling: one that carries signaling messages reliably between signaling entities, and another that interprets the messages. Signaling is often the most complex component of a computer network. Signaling is necessary for providing complex network services. Signaling has strict requirements for performance and reliability. RSVP (resource ReSerVation Protocol) is a kind of signaling protocol [29].

2.9 Network Pricing

Network pricing is how much a public network should charge for its services. Suppose we claim that a network provider can infer users' utilities from their willingness to pay for services. The idea is that the more utility a user obtains from using the network, the higher the price he is willing to pay. Thus, the network could charge different prices for different services, and users' willingness to pay this price would reveal their utility functions. Image that in the real world, you can drive a car to get to the destination by highway or by local road, you can save time if you choose highway, but you have to pay more. The key point is that by setting a price for usage, the network can control user demand, at least broadly, thus modifying the traffic load on the system. Therefore, pricing can be used as a tool for traffic control [30].

2.9.1 Peak-load Pricing

Traffic exhibits strong cyclical behavior at the time scale of a day and at the time scale of a week. In fact, operators look for traffic anomalies simply by overlaying traffic measured a week earlier over the current measurement. During the day, traffic peaks from 9am to 5pm, reflecting the working day. There is a typically a drop at lunchtime and dinnertime. However, it picks up again around 11pm, when telephone rates and Internet usage rates become lower, thus allowing users to save on tolls. This shifted peak is the result of peak-load pricing, which is a traffic control mechanism operating at the time scale of a day. Peak-load pricing shifts some user demand from the peak time to off-peak time, decreasing the peak load [18]. With peak-load pricing, the network charges more during peak hours, and less during off-peak hours. Some customers cannot wait until the off-peak hours, and they thus pay more. However, some customers can wait, and their demand is shifted to off-peak hours. Thus, peak-load pricing allows the network provider to deliver more utility to its customers, because overloading is reduced. In the future, with intelligent endpoints, sophisticated peak-load pricing scheme may become more popular [6].

2.9.2 Re-negotiation

Recall that a guaranteed-service connection must specify its traffic descriptor at the time of connection establishment. However, it is often impossible to a priori determine satisfactory traffic descriptors a priori. The application designer or application user can only guess the expected average rate of the application. If the guess is too high, then the user pays an unnecessarily high fee for its service, because the network must reserve resources for at least the user's declared average rate. If the guess is too low, the policer drops excess traffic, so that the received quality degrades. Sometimes, finding an adequate descriptor is hard even if we know the entire source behavior in advance. But if the application can renegotiate its traffic descriptor, these problems can be solved.

If a source can renegotiate its traffic descriptor at the beginning and end of every burst, its effective reserved rate is identical to its long-term average rate. However, this imposes a heavy signaling load on the network. Keeping worst-case delay and loss rate fixed, as the renegotiation frequency decreases, the effective reserved rate moves farther away from the average rate and approaches the source's peak rate. With stored traffic, the series of renegotiation points and renegotiation values can be pre-computed. Even for online interactive traffic, the application can observe past behavior and use this to predict future behavior. Thus, renegotiation does not pose a severe burden on applications. It does increase the network signaling load, and a user must trade-off between renegotiation frequency and the degree to which the effective reserved rate approaches the true long-term average rate [31].

2.10 Admission Control and Measurement-based Admission Control

When a connection is requested with its traffic descriptors and QoS requirements, the network decides whether to accept or reject the connection. The network determines if it has the necessary resources available to meet the requirements of the new connection while maintaining those of the ongoing connections [32].

The signaling network carries signaling messages and makes resource reservations. However, before a router controller can make these reservations, the admission control algorithm checks whether admitting the packet would reduce the service quality of existing packets, or whether the incoming packet's QoS requirements cannot be met. This decision depends on the choice of scheduling disciplines and the set of services provided by the network. If either of these conditions holds, the packet is either delayed until resources are available, or rejected. Admission control plays a crucial role in ensuring that a network meets its QoS requirements.

Measurement-based admission control allows us to deal with traffic sources that do not describe themselves. The idea is to admit packets based on a nominal description, but then to measure actual source behavior to automatically construct an appropriate descriptor. The danger with measurement-based admission control is that it assumes that past measurements of the system are a good indication of future behavior. The.hope is that with enough packets, a switch's load will change only very slowly compared with the number of packets arriving and leaving the network. Thus, even if the controller admits too many packets, it can simply deny admission to future packets, so that as some packets leave, the remaining packets receive adequate service quality.

Measurement-based admission control is particularly well suited for the controlled-load service model [21]. Recall that in this service model, the network guarantees a connection a nominal delay bound, but the connection's packets may still suffer deviations from this bound. If the connections behave similarly in the future, the delay bound will continue to hold. Because control-load service applications are willing to tolerate some packets with excessive delays, the measurement-based admission control algorithm can make some errors without aggravating customers. Measurement-based admission control is also necessary when sources can renegotiate their resource allocation. When a source sets up a packet, it may not know its future renegotiations. Thus, the admission control algorithm must guess, based on past behavior, whether or not to admit the packet [6].

In this chapter, we discussed some traffic control methods and components, all of them will be used to implement DiffServ in the Internet in Chapter 4. Traffic control is a necessary requirement for achieving a high QoS. If the traffic is controlled well, a good QoS is easily attainable. Bandwidth management is another important method to support QoS, which we will discuss in the next chapter.

Chapter 3 Bandwidth Management and its Methods

In this chapter we discuss network bandwidth management. The bandwidth is always finite and is an important system resource. Cheap and abundant bandwidth may be available in the future [33], but at present, we have to manage the bandwidth to use it efficiently. Bandwidth management is also an important way to ensure the QoS. We believe that there is a tight relationship between bandwidth and QoS, as it is easier to get good QoS if the bandwidth is adequate. A combination of bandwidth management and traffic control ensures a satisfactory level of QoS for the customer.

3.1 Bandwidth Management

There are four key areas of bandwidth management: bandwidth on demand, bandwidth aggregation, bandwidth augmentation, and switchover [34].

1) Bandwidth on demand

Bandwidth on demand means bandwidth is available when it is needed and charges are only incurred when data is actually being transmitted over the line. With bandwidth on demand, a connection is opened only when there is data to send and it is then closed as soon as the data has been sent. This process is totally transparent to users on the network. For example, when users are running a Web browser to access a remote Web server via ISDN, they cause an ISDN connection to be opened at the point of first access to the Web. While they are reading the data they have received, the connection times out because no data are being sent or received. As soon as they access the next page of information, the connection is re-opened. Since making the ISDN connection is so rapid, the users appear to have been connected all the time. The time-out parameters are usually configurable on the ISDN access devices and the most suitable values will depend on carrier tariff policy and the applications being used.

2) Bandwidth aggregation

Combining the bandwidth of two or more channels of the same type, on the same interface or across interfaces, is termed aggregation. In this situation, when a router receives the first packet for transmission, a channel is opened to the remote router. A further channel is then dynamically opened when the number of packets or bytes queued exceeds a certain value, which is normally user-defined. After each new channel is opened, there is a short delay before a subsequent channel is opened, allowing the existing queue to be emptied. When the measured data throughput indicates that fewer channels are needed, data are no longer transmitted on the channel that was opened last. If both ends stop sending data, the channel is closed after a user-specified time-out. This latency is used to accommodate bursty traffic patterns.

3) Bandwidth augmentation

Channels from different interfaces can also be combined. For instance, one channel on an interface is specified as primary while another is specified as secondary. Channels on the primary interface are used before channels from the secondary interface. This technique is used to combine bandwidth from interfaces of similar speed. Adding bandwidth from a different type of interface is known as augmentation. For example, using an ISDN B channel as on-demand bandwidth for a leased line is a common application of combined bandwidth. This allows a 64Kbps leased line to be used for average load, while an ISDN B channel is added when the leased line is saturated.

4) Switchover

Switchover enables traffic to be moved from one circuit to another, depending upon the traffic rate. A slow-speed leased line running at 19.2Kbps can be linked to a 64Kbps ISDN B channel. When the traffic rate on the leased line reaches saturation, the ISDN link is opened and traffic moved to it. Once the traffic rate drops below that of the leased line, the ISDN link is closed down and traffic diverted back to the leased line. The threshold at which traffic switches can be defined by the user. Switchover ensures that the most cost-effective circuit is always used, and provides a very cost-effective solution for networks with changing bandwidth needs throughout the day.

3.2 Some Methods for Bandwidth Management

Bandwidth is limited, but the requirement of bandwidth is not. Bandwidth management involves deciding what traffic has the highest priority, ensuring that it gets the bandwidth it needs, and deciding how to handle the lower-priority traffic. Bandwidth management ensures that network services are used only when required and closed down when there is no user data transmission [35]. This is critically important when services are being paid for, regardless of the amount of traffic being transmitted across the network. It also ensures that optimal services are used for particular applications and/or particular remote sites, and that extra bandwidth can be made available when there are unexpected bursts of traffic. Some methods for bandwidth management are addressed below.

3.2.1 Bandwidth Allocation and Dynamic Bandwidth Allocation

The system, made up of the users as well as the network, has various resources that can be used to meet service demands. However, in all realistic systems these resources are limited and some methods of allocating them is needed when total demand is greater than the resource limit. Bandwidth allocation is about efficiently allocating the network bandwidth among the sources.

Dynamic bandwidth allocation refers to techniques that allocate bandwidth according to instantaneous demand. For example, a typical TDM (Time Division Multiplexing) network would require separate allocations of bandwidth for the voice and data. Dynamic bandwidth techniques allow data to burst into the unused voice bandwidth, as it becomes available and force data to back off as voice connections are activated [36].

3.2.2 Bandwidth Sharing and Dynamic Bandwidth Sharing

The bandwidth sharing method relies on sharing the link bandwidth among a number of connections using one of the following methods:

1) Fair bandwidth sharing is based on sharing the link bandwidth among the different connections.
2) Bandwidth scheduling assigns a limited amount of bandwidth to a number of connections according to specific scheduling time slots.

Dynamic bandwidth sharing methods which rely on increased sharing of resources would yield better utilization of the network bandwidth. The bursty nature of data traffic could be exploited by allowing some users to consume the bandwidth during other users' idle periods.

3.2.3 Bandwidth Borrowing

If the whole bandwidth is assigned to all class of packets, each class is allocated a percentage of the bandwidth. When that limit is reached, normally no more traffic from that class can be forwarded. However, if the network link is not being fully used, a class can borrow bandwidth temporarily from its neighbor class, and send traffic at a percentage that exceeds its allocation. The configuration of a class defines the maximum percentage of bandwidth, including that borrowed, that can be used by a class at any time. Spare bandwidth is allocated temporarily to classes having the highest priority [34]. The proportion of the spare bandwidth given to a class depends on the percentage of bandwidth configured for the class. For example, suppose 20% of the available bandwidth is not being used, and there are three classes with packets queued. Two of the classes have priority 1, with bandwidths 1% and 9%, and the third class has priority 3 and bandwidth 12%. The priority 1 classes are given an additional 2% and 18% respectively, and the priority 3 class is not given any additional bandwidth. It is possible to define a class that has 0% bandwidth allocated but may borrow bandwidth from its parent class. A packet allocated to such a class is only forwarded if there is no other traffic of higher priority waiting. A class that has 0% bandwidth allocated is given borrowed bandwidth as though it had 1% bandwidth allocated. Allocating 0% and no borrowing to a class means that the class is blocked.

3.2.4 Bandwidth Reservation

Bandwidth reservation means that a request is made to the network to allocate a specific amount of bandwidth for data flow. It allows applications to reserve bandwidth and QoS along the data path. Many new content-rich applications, such as video conferencing, interactive multimedia video games or training programs, need stable, predictable QoS in terms of bandwidth and delay in order to function well. Bandwidth reservation protocol is based on the standard network control protocol RSVP (ReSerVation Protocol) [29], which allows Internet/intranet applications to reserve special QoS for their data. RSVP was proposed by the Internet Engineering Task Force (IETF), and is emerging as a standard protocol for bandwidth management. It is a component of the future Integrated Services (IntServ) in the Internet. When an RSVP-enabled multimedia application receives data for which it needs a certain QoS, it sends an RSVP request back along the data path, to the sending application. At each stage along the route, the QoS is negotiated with the routers or other network components. Non-RSVP network equipment simply ignores RSVP traffic and takes no part in the negotiation.

3.2.5 Preventing Bandwidth Starvation

Bandwidth can be controlled by simple mechanisms such as guarantees and limits. However, priorities provide the most powerful and flexible method to dynamically allocate limited bandwidth. The objective of priorities is to grant preferential privileges to one class of traffic over another. For example, a network manager could grant a higher bandwidth priority for Web traffic than SMTP traffic.

There are two types of bandwidth priorities: absolute and weighted. Absolute priority means to assign a priority level to each class of traffic. For example, if there are seven priority levels available for Internet traffic, Web traffic may be given a priority of 7, and SMTP traffic assigned a priority of 6. Absolute priority is inefficient because it operates on an all-or-nothing basis. When the line is oversubscribed, all higher priority traffic gets through before any lower priority traffic receives bandwidth. As a result, heavy Web usage may deny bandwidth to all SMTP connections. This situation is defined as bandwidth starvation. In order to avoid bandwidth starvation, we have to use weighted priority. Weighted priority allocates available bandwidth based on relative merit or importance. When using weighted priorities, each class of traffic is given a weight that is relative to all other weights defined in the management policy. The weights define the basis upon which traffic competes for available bandwidth. For example, Web traffic can be assigned a weighted priority of 60, and SMTP traffic can be given a weight of 20. When bandwidth resources are oversubscribed, the ratio of Web traffic to SMTP traffic is accurately maintained at a 60:20 ratio. Weighted priority provides the only mean to prioritize traffic and prevent starvation.

3.2.6 Bandwidth Pricing and Dynamic Bandwidth Pricing

The bandwidth allocated to a user is considered to be a commodity, which is sold by the network to the user. We view the users as placing a benefit, or willingness-to-pay, on the bandwidth they are allocated. Given a price per unit of bandwidth, a user's benefit function completely determines that user's traffic input. Users are assumed to act in their own best interests and to be capable of responding to changes in the price for bandwidth [13].

Assigning dynamic priority is difficult. If the real-time applications such as voice and video are given priority to ensure timely delivery, then data traffic may suffer higher loss though it may not be able to tolerate cell loss as well as voice. On the other hand, if priority is given to data and a lot of buffering is employed, then real-time applications may suffer large variable delays [30]. Hence we need a dynamic adaptive inter temporal priority scheme. The priorities should change to track changes in the network state or in the application requirements over multiple time periods. Rather than having a complicated priority scheme, a pricing scheme could be used. The operator would set the benefit functions for the different applications, and could also set different benefit functions for applications of the same type. Each application would then input traffic according to its assigned benefit function and the current state of the network, as reflected in the prices.

3.3 A Bandwidth Management Architecture for Internet

In this section we explain the general idea of Internet bandwidth management architecture. There are four entities in the architecture: nodes, hosts, applications and agents. The agents negotiate for bandwidth within the nodes and send their answers back to the hosts, which enforce the allocations on the applications. Hosts communicate through nodes in the interior of the Internet. These nodes have explicit knowledge of the characteristics of each connection through them, through negotiation with the host to set up connection. The nodes continually arbitrate and enforce maximum bandwidths for each connection.

This architecture ensures network fairness and makes it impossible for the network to become over-committed, since the nodes would keep the allocations below the limit of their capacity. Applications executing on hosts send agents to nodes [41].

3.3.1 Bandwidth Management Nodes

The management node combines two components, the Bandwidth Broker and routers. Routers include core router, boundary router, etc. These routers have different functions. The core router is for packet delivering, the boundary router is for packet shaping, marking, dropping, etc. The Bandwidth Broker is another important component, and is addressed below.

3.3.1.1 Bandwidth Broker: A Possible Solution for Bandwidth Allocation

Traditionally, the relation between a customer and the service provider is based on a fixed bandwidth, in which all traffic is handled in the same way (best-effort service). The recent popularity of the network has led to a shortage in network capacity. This can cause problems especially for the performance of mission critical applications. To solve this problem network service providers want to create new services, that guarantee the customer bandwidth, or at least a better than best-effort service. These guarantees are not always needed, and may be changed in the course of time.

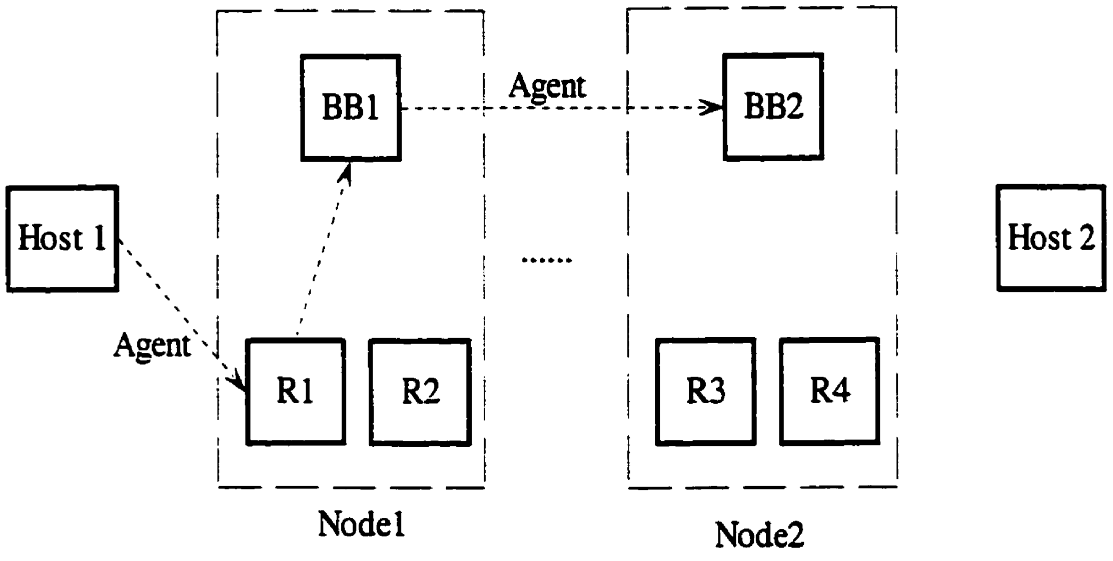
We propose the Bandwidth Broker architecture as a possible solution for the Internet bandwidth allocation [37]. The tasks a Bandwidth Broker can fulfill are numerous, but the main task is to negotiate a contract between the customer and the service provider, which sets the specifications (bandwidth, QoS, duration of contract, price, etc.) of a desired connection. The parameters of the Bandwidth Broker may vary depending on where the priorities of an application lie. An FTP connection would want high bandwidth, which may vary, and low package loss but does not care much about delay. An Internet Telephony connection would demand low delay, low jitter and a fixed bandwidth.

Since the management node contains Bandwidth Broker and routers, it has two main functions--arbitration and packet forwarding. The Bandwidth Broker is in charge of arbitration; the router is in charge of packet forwarding. They can be separated clearly.

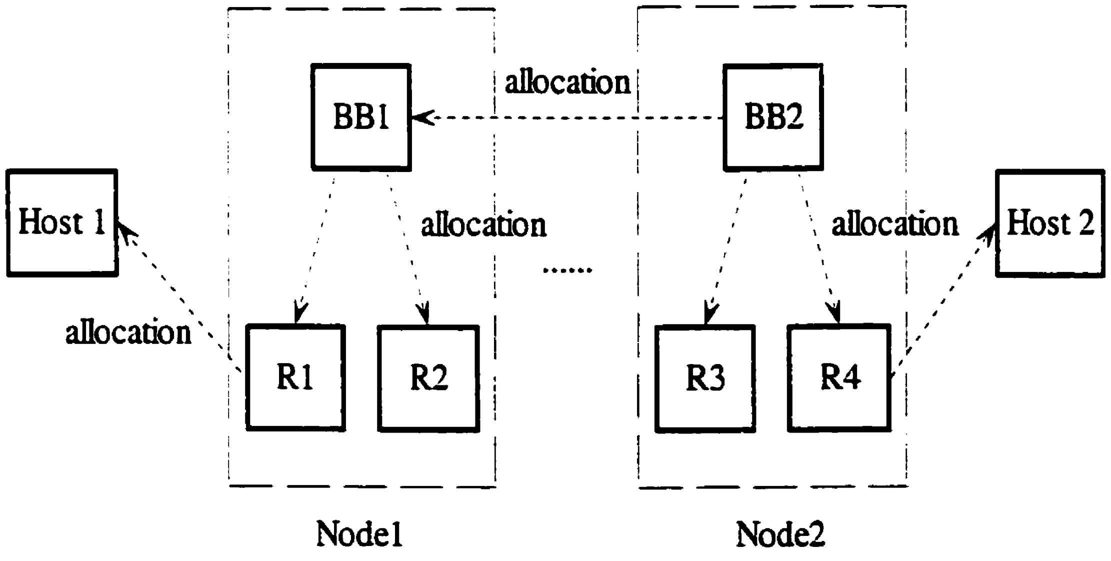
Arbitration is the process of determining the bandwidths allocated to each connection through the management node. Whenever the availability of bandwidth at the management node has changed sufficiently since the last negotiation round, the node determines the available resources and conducts a negotiation round, through which the applications communicate their desires for bandwidth and the management node sets their bandwidth allocations. Once the bandwidth allocations are set, a fair packet forwarding scheme can be used to pass packets along according to the allocations.

3.3.1.2 Management Nodes Assign Bandwidth

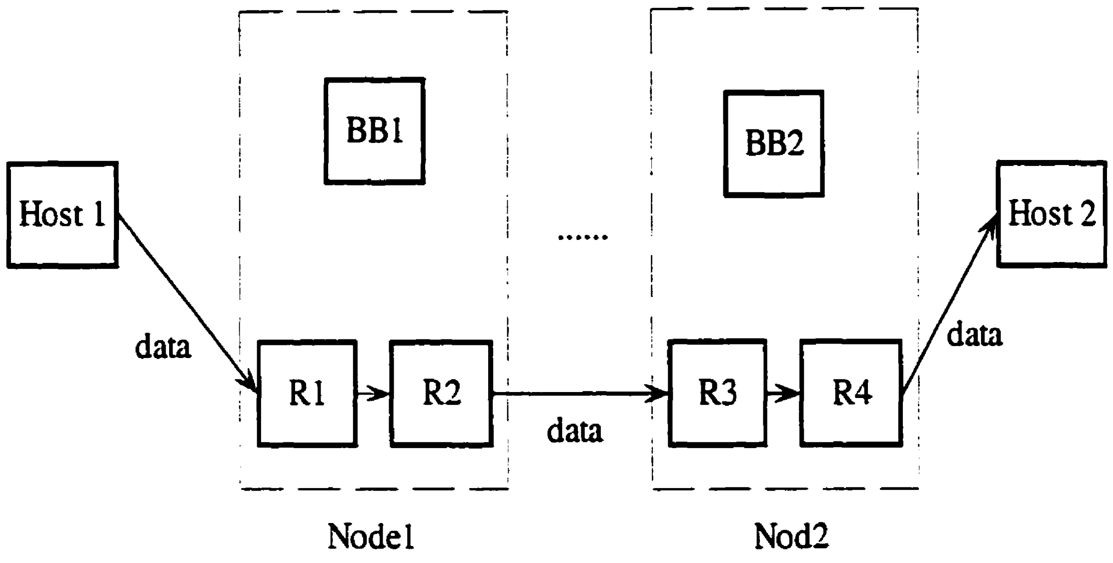
The management nodes assign bandwidth to connections based on agents that user applications send them. The application can be aware of what kinds of data rate tradeoffs are best for it, so it can compose an agent to negotiate for bandwidth on its behalf and send it to the management nodes along a connection. Each management node uses a bidding process to determine the amount of bandwidth each agent wants, after that, the bandwidth is assigned.



a) Host1 submits an agent to Bandwidth Brokers.



b) The Bandwidth Brokers allocate bandwidth at any time, send the results to the routers and back to the Host1.



c) Data flows through the routers according to the bandwidth allocations.

Figure 3.1 Management nodes assign bandwidth

- **Scenario**

The process of management nodes assigning bandwidth is shown in Figure 3.1. We assume that there are only two nodes. We propose the following scenario:

1. Host1 sends an agent of application to Node1, the application includes some connection parameters, such as bandwidth, delay, jitter, rate, price, etc. The agent is on behalf of the application to negotiate with the Bandwidth Broker. The agent gets the connection information from the application.
2. Router1 (R1) receives the agent, forwards it to the Bandwidth Broker1 (BB1). BB1 negotiates with the agent based on the current situation of traffic and bandwidth. The price is dynamic because the situation of traffic and bandwidth is variable.

3. Two or more agents apply for service at the same time, and bandwidth is not enough to satisfy all the connections, bidding for service happens. The agent who pays more wins, others are refused.
4. The request is accepted by BB1. BB1 forwards the agent to Bandwidth Broker2 (BB2). If the request is denied, an error message is sent back to Host 1 by the Agent.
5. BB2 negotiates with the Agent like step 2. If some agents are applying for service simultaneously and the bandwidth is not enough, then they bid for service like step 3. When the request is accepted, BB2 sets the connection and informs R3 and R4 of the classification and the policing rules. After that, BB2 sends the Agent to BB1 with a confirmed message. If the request is denied, an error message is sent back to Host1.
6. BB1 receives the Agent with a confirmed message from BB2, it sets the connection and inform R1 and R2 the classification and shaping rules. So, if the traffic of the admitted flow is non-conformant R1 will shape it. Then, BB1 sends back the Agent to Host 1 with a confirmed message.
7. Host1 receives the Agent with the confirmed message, it starts to transmit data.

This negotiation idea can be used for Differentiated Services (DiffServ) in the Internet. We will discuss it in greater detail in Chapter 5.

3.3.2 Agents

One factor that limits the responsiveness of any arbitration mechanism is the speed with which an application running on a local computer can communicate with management nodes within the network. Most networks do not have a direct-line topology, and there is overhead in forwarding packets. All of this adds up. Furthermore, some possible schemes for negotiation between the management nodes and the applications require several rounds of communication. All this time adds up and decreases the speed with which the network can adapt.

To counteract this, the applications could send agents to negotiate on their behalf. Agents are small programs that can run on remote machines. These agents would be propagated along each connection to all of the affected management nodes. Once at a node, they would be invoked by the arbitration process and respond as the application would, except without the round-trip delay of communicating with the application itself. An agent is an interpretable function that takes a number of inputs and produces a bid for bandwidth. The inputs depend on the negotiation scheme used by the architecture. An application that wishes to open a connection across the network would encapsulate the relevant information about the connection in an agent and send it onward to the nearest management node. These functions are taken in by a network node and are used to negotiate for resources on behalf of the application, remotely. The arbitration process determines allocations of bandwidth for all the connections through a particular management node, with the agents providing knowledge of the behavior and requirements of each application to the management node [39].

Mobile agents can be used if it is necessary.

3.3.3 Mobile Agents

Mobile agents are autonomous, intelligent programs that can migrate from machine to machine in a heterogeneous network. The program chooses when and where to migrate. It can suspend its execution at an arbitrary point, jump to another machine and resume execution on the new machine. From a computation point of view, mobile agents co-locate data and computation by bringing the computation to the data, rather than by bringing the data to the computation. Mobile agents have the necessary autonomy to make decisions, and to interact with other agents and services to accomplish their goals.

Mobile agents can reduce the network traffic. Most communication protocols involve several interactions, especially when security measures are enabled. This causes a lot of network traffic. With mobile agents, one can package up a conversation and ship it to a destination host where the interactions can take place locally [35,38]. Mobile agents can be used to build up tomorrow's intelligent Internet. But, in our model, the situation is simple as we just use the normal agent.

3.3.4 Negotiation Process

Once agents are installed in management nodes, they must cooperate with the arbitration mechanism in the node to determine allocations of bandwidth. The arbitration mechanism will use some sorts of negotiation methods that are both fair for all agents and difficult for an agent to subvert.

3.3.4.1 Negotiation and Agents

The negotiation process works as follows. The management node tracks a current price for bandwidth, which the agents buy from it. The node doles out a certain amount of credits per second to the agents, which they use to purchase bandwidth. Each host submits an agent to the ISP's management node. These agents take as inputs their last request for bandwidth and the current price of bandwidth, and return a new request for bandwidth. The management node cycles through the agents, asking them for their new requests for bandwidth and determines a new price for bandwidth based on their requests. Once the total bandwidth requests converge, the management node sends out the allocations to hosts [40].

3.3.4.2 Credit-based Bandwidth Allocation

Using a pricing method is an effective way to manage bandwidth allocation [13,41]. The main idea of a pricing method is for all applications to bid money for services. An auction of network bandwidth would presumably be the best possible way to ensure that applications do not attempt to grab all available bandwidth, since that would cost large amounts of money. In this scheme, great hurdles need to overcome, such as accurate billing, secure transactions, etc.

Within a single organization, like an intranet, some of these restrictions can be relaxed. So, we propose another method, called Credit-based method. In our method, bidding could be done with credits replacing money. Relative fairness can be assured by doling out credits on a regular basis, with more money going to higher-priority connections. This devolves to a weighted fair share algorithm if all agents can do is spend the money as they receive it – each connection gets a proportion of the bandwidth equal to its share of the entire pool of credits doled out at a time. However, if the agents have some ability to save credits, perhaps even to spend credits they don't have, then they can plan for the future. For instance, an application with quantized bandwidth requirements could save credits when forced to switch to a bandwidth step lower than its share, in anticipation of a time when it will be able to maintain a higher bandwidth step. In another instance, if there is some channel for agents to receive commands from the application, or even for agents to be replaced, an agent might hoard some credits, looking ahead to a time when the application needs to send data more urgently than it does now. The basic algorithm arrives at a price by stating a price to each agent and taking the resultant bandwidth requests and determining the bandwidth allocation. The price is then changed and the agents are invoked again. Stepwise refinement continues until the requests converge on a value that is mutually satisfactory to the agents and does not over allocate the outbound network link [39].

The algorithm is as follows:

```
 do
   for each agent
           bandwidth allocation[agent] = agent (price, credit.balance)
   bandwidth.allocations = summation of bandwidth.allocation[1..n]
   allocation.ratio = bandwidth.allocations / bandwidth.available
   price = price * allocation.ratio
until (allocation.ratio converges on 1)
```

Since the amount of credits agents have to spend is limited, and all agents receive the same amount of credits, the price of bandwidth must always be finite. The algorithm will converge as long as the minimum balance the agents can negotiate sums to less than the available bandwidth, although if bidding forces prices unreasonably high it may take longer. When the agents are forcing the price of bandwidth high temporarily, they are also spending their allocations of credits very quickly, and will not be allowed to spend more credits than they have. This is effective at ensuring sane negotiations. Agents that do not have bandwidth prices above which they are not willing to buy any bandwidth at all could be considered incorrect. If, however, the minimum bandwidth needs of all the agents at an arbitrator add up to more bandwidth than is available, the management node is over-committed. Some other mechanism must be used in this case to police the allocations and return the system from an over-committed state. Since this case only occurs in the case of variability-intolerant applications or very poorly written or actively malicious agents, we would like this mechanism to determine which agents are most at fault and deny them service. We would also like to use the amount of money given to this agent as a criterion – if an application is particularly important, human intervention to provide it with more resources before it runs out should ensure that it is not capriciously killed. So, some combination of credit allocations and observed adaptiveness of the agents should be a workable method of policing this unfortunate case.

3.3.4.3 Enforcement of Allocations

Queues, packet scheduling and packet dropping are use to enforce allocations.

- Queues

The management node will keep a separate packet queue for every connection it handles. Each of these queues will be of some reasonable length – sufficient to store enough packets to smooth out any unwarranted variations in the network, while short enough that applications counting on low-latency connections are not unduly affected.

- Packet Scheduling

Packets will be removed from the queues as the network permits, using a fair scheduling algorithm, such as Virtual Clock or Weighted Fair Queuing to ensure that the allocations are obeyed [33,42]. By accepting packets, placing them into queues and then draining the queues in a priority-based fair manner, it ensures that all data leaving the management node abides by the allocations, and thus enforces the allocations.

- Packet Dropping

The queues have a finite length, and if an application's queue is filled faster than it drains for a long enough period, packets are dropped. This acts to penalize applications for sending too much data, providing their authors incentive to remain within their allocations.

In this chapter, we propose an adaptive bandwidth management architecture, which allows Internet users to transmit data of different speeds at different prices. This idea may be expanded to differentiated pricing for differentiate services in the Internet. In Chapter 4 we present the main concepts behind differentiated services in Internet.

Chapter 4 Differentiated Services in Internet

Differentiated Services is a multiple service model that can satisfy different QoS requirements, and is based on the principle "*pay more, get more*". The network should provide customers with different QoS based on their different levels of payment. Today, the Internet hosts a wide range of applications and user applications with different requirements. If the network were able to offer proper QoS for all applications, both the amount of services and users would be higher.

4.1 Generalized and Specialized Differentiated Services of Networks

There are many kinds of networks in the world, but there is no generally accepted taxonomy into which all computer networks fit. Computer networks can be classified based on several factors. for example, bandwidth. common applications, common hardware, etc. An alternative criterion for classifying networks is their physical size. Distance is important as a classification metric because different techniques are used at different scale. We give a classification example in the Table 4.1 [1]:

Interprocessor distance	Processors located in same	Example
0.1 m	Circuit Board	Data flow machine
1 m	System	Multiprocessor
10 m	Room	Local area network
100 m	Building	Local area network
1 km	Campus	Local area network
10 km	City	Metropolitan area network
100 km	Country	Wide area network
1,000 km	Continent	Wide area network
10,000 km	Planet	The internet

Table 4.1 The classification of networks

Some important networks [1]:

- LANs (Local Area Networks), for example, a computer network in a company's department, such as Ethernet network;
- MANs (Metropolitan Area Networks), for example, a cable television network within a city and FDDI network;
- WANs (Wide Area Networks), for example, an ISDN network;
- internet, for example, the well-known worldwide Internet.

4.1.1 A Generalized Differentiated Services (GDS) Network Model

We can imagine that all currently existing networks are all in this GDS network model, where there are a great many different users, different tasks, and many different networks providing tremendous services. The GDS network model is a virtual network model, but it gives the idea of Generalized Differentiated Services and Specialized Differentiated Services.

4.1.2 Generalized Differentiated Services (GDS)

The GDS model of networks is a general idea, but for more specificity it can be divided into Hard Differentiated Services (Hard DS) and Soft Differentiated Services (Soft DS). Hard DS is based on different network hardware, such as Ethernet and Token Ring. Soft DS is based on different network software. Protocol is the most important network software. There are a lot of protocols, like TCP/IP, ATM protocol, etc.

4.1.3 Specialized Differentiated Services (SDS)

Here, SDS is used just for IP (Internet Protocol). Because IP is the most important data transport protocol, it is supported widely. IP networks are based on Internet Protocol. Internet is the largest IP network, which is a worldwide collection of computer networks. IP provides a connectionless, unreliable, best-effort packet delivery system.

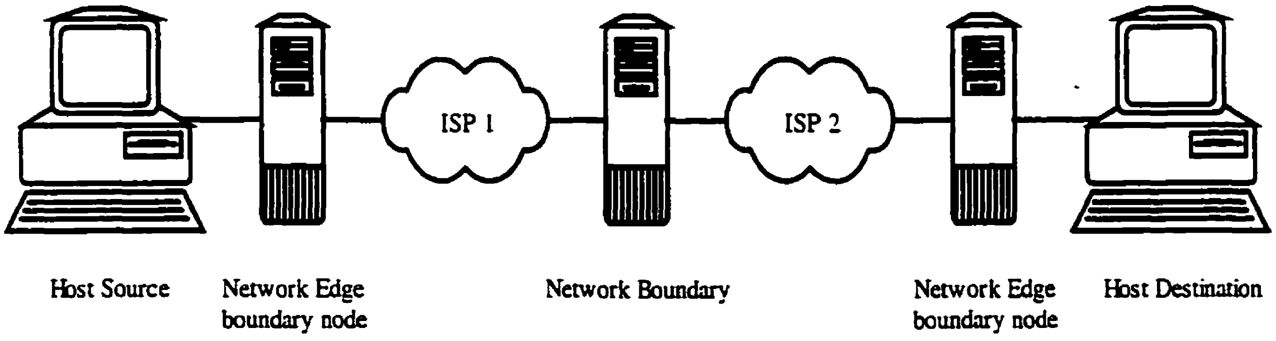
4.2 Differentiated Services in Internet

Differentiated Service (DiffServ) has been developed by the Internet Engineering Task Force (IETF), which is the first step for QoS in the Internet. SDS is the same as DiffServ. The Internet is so important and popular, it has an enormous amount of users in the world. But today's Internet can only provide best-effort service, and it is not able to offer proper QoS to meet all needs. IETF defines several kinds of QoS for the Internet, such as Differentiated Services (DiffServ) [43], Integrated Services (IntServ) [44,45], Multi-Protocol Label Switching (MPLS) [46,47,48], etc.

Here, we only focus on DiffServ, because it is easy to be implemented.

4.2.1 General Architecture of DiffServ

Network edge and network boundary are important concepts in DiffServ. Network boundary is basically a router which links two network clouds. Network edge is a particular boundary node, which resides at the edge of the whole DiffServ-compliant area [49]. The architecture of DiffServ is shown in Figure 4.1.



ISP (Internet Service Provider)

Figure 4.1 DiffServ's General Architecture

The boundary nodes evaluate and set the bits in the Differentiated Service byte (DS byte) for each packet and condition the packets based on preinstalled service profiles [50]. The profiles are set by the operators according to the contracts with their customers. DS byte is used to determine how the packets are treated. The treatment, called Per-Hop-Behavior (PHB) or Behavior Aggregate (BA), can include different priorities involving the queuing delay, different priorities in the drop decisions if the queues overflow, route selection, etc. At the boundaries, packets are classified using any information in the packet headers, for example, IP addresses and port numbers. The classification and the profiles can be as simple or as complicated as desired. In the core network, only the DS byte needs to be investigated, which simplifies the classification [51].

This architecture is used, because:

1) Sophisticated classification, marking, policing and shaping operations are only needed at boundary of the networks. ISP core routers only need to implement Behavior Aggregate (BA) classification. Therefore, it is easier to implement and deploy DiffServ.
2) ISP networks usually consists of boundary routers connected to customers, and core routers/switches interconnecting the boundary routers. Core routers must forward packets very rapidly and therefore must be simple. Boundary routers need not forward packets very rapidly because customer links are relatively slow. Therefore, they can spend more time on sophisticated classification, policing and shaping.

4.2.2 Related Control for Supporting DiffServ

We need some related controls for supporting the implementation of DiffServ, such as traffic control, bandwidth and queue management, etc.

4.2.2.1 Traffic Control for DiffServ in Boundary Routers

Traffic control is usually performed at the boundary routers and it consists of four processes: classification, marking, policing and shaping. For each traffic flow through the boundary, the router only performs either policing or shaping [52]. Some traffic control operations are as follows.

- **Admission Control**

This process is to decide whether to accept a request for resources.

- **Classification**

The process of sorting packets is based on the content of packet headers according to defined rules. Classification is done for matching packet headers against entries in the classifier table. Every packet is classified to a class. After the classification, the packets within a particular class receive similar treatment, while the treatment can vary between different classes. Treatment is composed of marking, policing, shaping, scheduling, etc.

In the DiffServ boundaries, classification can be based on any combination of packet header fields. In IPv4, the fields in the headers that are meaningful in the classification are the source and destination IP addresses, protocols, such as UDP, TCP, ICMP, etc., and the source and destination port numbers in UDP and TCP. If only IP addresses are used, the network can provide DiffServ on host or sub-network level. If application level differentiation is required, the classification has to take the port numbers into account. Some of the port numbers are well known, but IP telephony (or H.323) uses dynamic port numbers. In that case, the application would have to signal its port numbers dynamically to the DiffServ edge. Of course, the host can do DS marking by itself and thus avoid the problem. In IPv6, there is also the flow label field, which is applicable. In DiffServ, the point is that complex classification is needed only at the boundaries, otherwise, only the DS-byte is used. In other words, DiffServ aggregates the classifier's state in the core network [50].

- **Behavior Aggregate (BA) Classification**

BA Classification is the process of sorting packets based only on the contents of the Differentiated Service field (DS field). The DS field is the field in which the Differentiated Services class is encoded. It is the Type of Service octet in the IPv4 header or the Traffic Class octet in the IPv6 header [53].

- **Multi-Field (MF) Classification**

The process of classifying packets based on the content of multiple fields such as source address, destination address, TOS byte, protocol ID, source port number, and destination port number.

- **Marking**

Marking is the process of setting the DS field in a packet at the network boundaries. Marking can be performed by the application, the operating system or the edge router. Marking gives each packet a particular PHB, which determines the treatment the packet gets in the core routers. Marking is usually performed according to the results of either policing or shaping.

- **Shaping**

Shaping is the process of delaying packets within traffic stream to conform it to some defined traffic profile. Shaping causes the packet stream to be conformed to some configured traffic properties. Shaping is often based on the leaky bucket algorithm. The shaper smoothes the bursts of a stream, but delays non-conforming packets.

- **Policing**

Policing is the process of handling out-of-profile traffic, for example, discarding excess packets. Policing monitors the packet stream based on its profile. A simple policer is implemented using the token bucket algorithm [28], which characterizes the packet stream with two parameters: average rate and burst size. For each packet of a stream, the policer declares whether the packet was conformant or non-conformant to the stream's profile.

4.2.2.2 Traffic Control in Core Router based on PHB

The purpose of the PHB is that the packets marked with different PHB values should experience differing service in the core routers. There are several ways for a router to implement differing service, but the most important mechanisms are scheduling and queue management.

A) Scheduling

Scheduling is the process of deciding which packet to send first in a system of multiple queues. In general, schedulers can be characterized as work-conserving or non-work-conserving. A scheduler is work-conserving if it is never idle when a packet is queued in the buffer. Non-work-conserving server may, for example, postpone the transmission of a packet when it expects a higher-priority packet to arrive soon, even though it is currently idle.

Some scheduling algorithms:

1. Priority Queuing

Priority Queue is a simple scheduling algorithm [30]. The queues are arranged in strict priority order, and a particular queue gets service only if there are no packets in the higher priority queues. Priority queuing can guarantee small delay for the highest class, but the other classes face a possible starvation, if the higher classes use all the available bandwidth.

2. Weighted Fair Queuing (WFQ)

WFQ is a representative example of a work-conserving priority-based scheduler [30]. If the weights in WFQ corresponding to the individual queues are equal, the algorithms divide the capacity of the output link by emulating a time-division multiplexer (TDM). If the weights are not equal, the queues share the capacity according to their weights. If any of the queues does not have enough packets to send out, the other queues share its portion according to their weights.

3. Weighted Round Robin (WRR)

WRR is a good example of work-conserving frame-based scheduler [30]. WRR serves each queue in a round-robin fashion, and for each turn, a number of bits corresponding to the queue's weight is "pulled out" from the queue. Thus the link capacity is divided according to the weights as in WFQ. In a worst-case situation, a packet arrives to a queue just after the queues turn. In that case, the maximum queuing delay will be the sum of the weights of all other queues, if all the other queues have also enough packets. In that sense, WRR is not as ideal as WFQ, but it is simple to implement. This may become a deciding factor, if the link speeds increase faster than the pure processing power does.

Using different queues, the network operator can differentiate the service experienced by differing PHBs.

B) Queue management

Queue management controls the length of packet queues by dropping packets when necessary or appropriate [46]. In DiffServ the idea is that the dropping decisions take the PHB values into account. Different PHBs can be treated as different drop preferences. The usual mechanism is that the router constantly measures the length of its queues and sets dropping thresholds based on the measurements [52].

If different PHB values translate into differing drop preferences, their service differs dramatically during congestion. For TCP traffic, this is seen to the users as differing throughput, because TCP slows down when packets get dropped. For UDP traffic, the effect is naturally differing packet loss ratio, which can be important for example with streamed video service.

One of the most popular queue management algorithms is Random Early Detection with In and Out (RIO) [54].

RED (Random Early Detection) [55] is a queue management scheme that drops packets randomly. This will trigger the TCP flow control mechanisms at different end hosts to reduce send rates at different time. By doing so, RED can prevent the queue at the routers from overflowing, and therefore avoid the tail-drop behavior (dropping all subsequent packets when a queue overflows). Tail-drop triggers multiple TCP flows to decrease and later increase their rates simultaneously. It causes network utilization to oscillate and can hurt performance significantly. RED has been proved to be useful and has been widely deployed.

RIO is an advanced RED scheme, it maintains two RED algorithms, one for *in* packets and the other one for *out* packets. RIO use two thresholds to drop packets, the first one is for *out* packet, the second is for *in* packet. When the queue's capacity exceeds the first threshold, the *out* packet will be dropped. The *in* packet will be dropped only when the second threshold is reached.

4.3 Implement DiffServ based on Two-bit Architecture

The two-bit (premium bit P, assured bit A) DiffServ architecture indicates three traffic classes 00=best effort, 10=premium, 01=assured. The premium class is targeted for real-time traffic, whereas the assured class receives better than best-effort treatment subject to drop probability, and is thus suitable for TCP. At the edge router, the packets are classified and premium and assured flows are set. Premium flows are shaped to constant bit rate and they are marked with 10. The bucket is very shallow, and overflow packets are discarded. Assured traffic is subject to token bucket policer, and conformant packets are marked with 01, while non-conformant packets are marked with 00. In the core routers, Premium bit is used to classify the packets into two queues. Premium traffic goes into the upper queue, which always has a strict priority over the lower queue. In the lower queue, a RIO is run based on the assured bit. Thus the assured traffic has lower drop probability than the best-effort traffic. The two-bit architecture is shown in Figure 4.2.

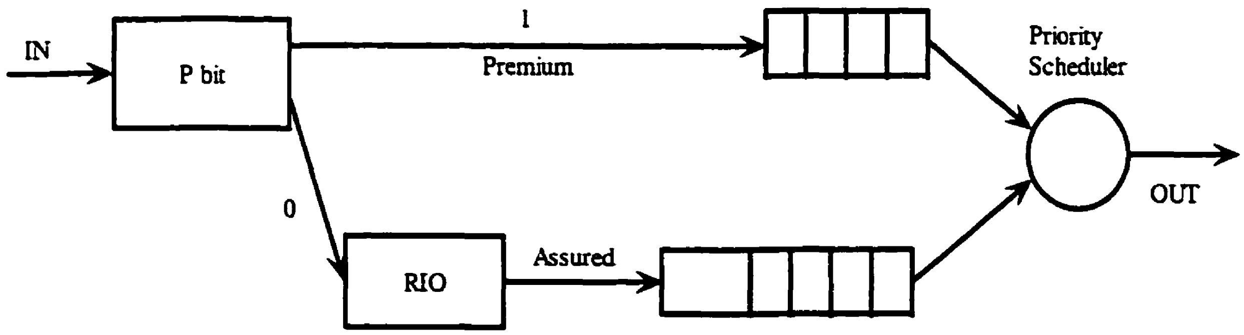


Figure 4.2 Two-bit DiffServ architecture

4.3.1 Service Level Agreement (SLA)

In order for a customer to receive DiffServ from their Internet Service Provider (ISP), the customer must have a Service Level Agreement (SLA) with its ISP. SLA is a service contract between a customer and an ISP. SLA specifies the forwarding service a customer should receive. A customer may be a user organization or another provider domain. A SLA basically specifies the service classes supported and the amount of traffic allowed in each class. A SLA can be static or dynamic. Static SLAs are negotiated on a regular basis, e.g. monthly and yearly. Customers with Dynamic SLAs must use a signaling protocol, e.g. RSVP, to request for services on demand. Customers can mark DS fields of individual packets to indicate the desired service or have them marked by the leaf router based on MF classification [51].

At the point of ingress to the ISP networks, packets are classified, policed and possibly shaped. The classification, policing and shaping rules used at the ingress routers are derived from the SLAs. The amount of buffering space needed for these operations is also derived from the SLAs. When a packet transmits from one domain to another, the SLA between the two domains will determine whether to re-mark its DS field.

4.3.2 Premium Service Implementation

The proposal of Premium Service was made by Van Jacobson [56]. Premium Service provides low-delay and low-jitter service for customers that generate fixed peak bit-rate traffic. Each customer will have a SLA with its ISP. The SLA specifies a desired peak bit-rate for a specific flow or an aggregation of flows. The customer is responsible for not exceeding the peak rate. Otherwise, excess traffic will be dropped. The ISP guarantees that the contracted bandwidth will be available when the traffic is sent. Premium Service is suitable for Internet Telephony, Video Conferencing, etc. [46] Because Premium Service is more expensive than Assured Service, it is desirable for ISPs to support both static SLAs and dynamic SLAs. Dynamic SLAs allow customers to request for Premium Service on demand without subscribing to it. Admission control is needed for dynamic SLAs. Premium Service can be implemented as follows.

At the customer side, some entities will decide which application flow can use Premium Service. The leaf routers connected directly to the senders will do MF classifications and shape the traffic. We can consider that there is a P-bit in the DS field. If the P-bit of a packet is set, this packet belongs to the premium class. Otherwise, the packet belongs to the Assured Service class or best-effort class. After the shaping, the P-bits of all packets are set for the flow that is allowed to use Premium Service. The exit routers of the customer domain may need to reshape the traffic to make sure that the traffic does not exceed the peak rate specified by the SLA. At the provider side, the ingress routers will police the traffic. Excess traffic is dropped. All packets with the P-bit set enter a Premium Queue. Packets in the Premium Queue will be sent before packets in the Assured Queue. Firstly, by admission control, the total amount of premium traffic can be limited to a small percentage, say 5%, of the total traffic. Secondly, excess packets are dropped at the ingress routers of the networks. Non-conformant flows cannot impact the performance of conformant flows. Thirdly, premium packets are forwarded before packets of other classes, they can potentially use 100% of the link bandwidth.

Therefore, if premium traffic is distributed evenly, these three factors should guarantee that the service rate of the Premium Queue is much higher than the arrival rate. Therefore, arriving premium packets should find the Premium Queue empty or very short most of the time. The delay or jitter experienced by premium packets should be very low. By limiting the total amount of bandwidth requested by Premium traffic, we use Weight Fair Queuing (WFQ) between the Premium Queue and the Assured Queue to guarantee that premium traffic will not starve the assured and best-effort traffic.

4.3.3 Assured Service Implementation

The proposal of Assured Service was made by Kathleen Nichols [51]. Assured Service is intended for customers that need reliable services from their service providers, even in time of network congestion. Customers will have SLAs with their ISPs. The SLAs will specify the amount of bandwidth allocated for the customers. Customers are responsible for deciding how their applications share that amount of bandwidth. SLAs for Assured

Service are usually static, that means the customers can start data transmission whenever they want without signaling their ISPs. Assured Service can be implemented as follows.

Firstly, classification and policing are done at the ingress routers of the ISP networks. If the Assured Service traffic does not exceed the bit-rate specified by the SLA, they are considered as *in* profile. Profile is a description of properties of a traffic stream such as rate and burst size. Otherwise, the excess packets are considered as *out* of profile. Secondly, all packets, *in* and *out*, are put into an Assured Queue to avoid out of order delivery. Thirdly, the queue is managed by a queue management scheme called RIO (RED with In and Out).

4.3.4 Two-bit DiffServ Implementation

Combining the implementations of Premium Service and Assured Service, we can implement the two-bit DiffServ as follows.

1) Customers negotiate SLAs with ISPs. The SLAs specify what services the customers will receive. SLAs can be static or dynamic. For static SLAs, customers can transmit data at any time. For dynamic SLAs, customers must use a signaling protocol such as RSVP to request for services on demand before transmitting data. The Bandwidth Brokers (BB) in the customer domains decide how applications share the services specified by the SLAs. The DS fields of packets are marked accordingly to indicate the desired services.
2) The ingress routers of ISPs are configured with classification, policing and re-marking rules. The egress routers of ISP networks are configured with re-shaping rules. Such rules may be configured manually by network administrators or dynamically by some protocol such as RSVP [45]. ISPs must implement admission control in order to support dynamic SLAs. Classification, marking, policing and shaping/reshaping are only done at the boundary routers. Core routers are shielded from the signaling process. They need only implement two queues with strict priority. They process packets based solely on their DS fields.

4.3.5 Implementation Performance

There are two parts of implementation performance: (a) customer performance, and (b) ISP performance.

A) Customer performance

Given a SLA, a customer domain should decide how its hosts share the services specified by the SLA. This is customer performance. There are basically two choices.

1) Each host makes its own decision to use the service.
2) A resource controller called QoS Broker or Bandwidth Broker (BB) makes decision for all hosts [3,8]. A Broker can be a host, a router or a software process on an exit router. It is configured with the organizational policies and it manages the resources of a domain. A domain may also have backup Brokers. Since all hosts must cooperate to share a limited amount of resources specified by the SLA, it is technically better to have a Broker to allocate resources.

At the initial deployment stage, hosts may send their packets unmarked. The exit routers mark them before sending them out to the ISPs. The packets are treated as best-effort traffic inside the customer domain. In later deployment stages, when a host wants to send traffic, it will consult the Broker for a service type. The Broker decides the service class and replies to the sender. For premium traffic, the Broker will then use some protocols, e.g., RSVP, to set the classification, marking and shaping rules at the leaf router that is directly connected to the sender [57]. The Broker may also set the reshaping rules at the exit router. Senders will send their packets unmarked and the leaf routers will mark them. If the SLA between a customer and its ISP is dynamic, the Broker in the customer domain must also use some signaling protocol to request resources on demand from its ISP.

B) ISP performance

Given the SLAs, ISPs must decide how to configure their boundary routers so that they will know how to handle the incoming traffic, this is ISP performance. For static SLAs, boundary routers can be manually configured with the classification, policing and shaping rules. Resources are therefore statically allocated for each customer. Unused resources can be shared by other customers. For a dynamic SLA, resource allocation is closely related to the signaling process. The Broker in the customer domain uses RSVP to request for resources from its ISP. At the ISP side, the admission control decisions can be made in a distributed manner by the boundary routers or by a Broker. If boundary routers are directly involved in the signaling process, they are configured with the corresponding classification, policing and shaping rules when they grant a request. If a Broker is involved rather than the boundary routers, then the Broker must configure the boundary routers when it grants a request. In both cases, the ISP core routers must be shielded from the requests to avoid the scalability problem.

Both customer and ISP need routers' support to finish their performances. Routers' support is as follows.

1. The leaf routers in customer domains need to implement MF classifications, marking, and shaping.
2. The ISP ingress routers need to implement policing and re-marking.
3. The ISP egress routers need optionally to implement re-shaping.
4. All routers need to implement BA classification and two queues with strict priority.
5. If dynamic SLAs are supported, each customer domain will need a Broker to request for service on behalf of the domain and to allocate services inside the domain. Signaling and admission control mechanisms are needed in both customer domains and ISP domains.

DiffServ can be implemented based on two-bit architecture, but the architecture has some drawbacks. We need to improve that, the methods will be discussed in Chapter 5.

4.4 Coordinated Control

Coordinated control can be broken down into traffic control, bandwidth control and queue control. These control methods have been mentioned in Chapter 2 and Chapter 3. In order to get high QoS, we need the coordinated control, because coordinated control can avoid congestion, lighten and balance the traffic load. Coordinated control is the process of arranging how traffic flows through the network so that congestion caused by uneven network utilization can be avoided [66,75].

There are several kinds of QoS services in the Internet, such as Integrated Service (IntServ), DiffServ, MPLS, etc., but actually, there is little difference when the traffic load is light. So, it is necessary to do coordinated control in the first place.

The main aims for coordinated control are as follows.

- Traffic control: avoid congestion by congestion control, reasonable routing, load balancing, etc.
- Bandwidth control: supports traffic control by increase the utilization of bandwidth.
- Queue control: supports traffic control by appropriated queue algorithms, such as RIO, WFQ, etc.

4.4.1 Traffic Control

In DiffServ, where the goal of traffic control is to avoid congestion, some methods can be used, such as congestion control, the reasonable routing algorithm and the traffic load balancing algorithm [77,79]. The methods of congestion control have been mentioned in Chapter 2, here, we describe the reasonable routing algorithm.

Usually some parts of the network are overloaded while other parts are lightly loaded. Uneven traffic distribution can be caused by the current Dynamic Routing protocols such as RIP (Routing Information Protocol), OSPF (Open Shortest Path First) and IS-IS (Intermediate System-to-Intermediate System), because they always select the shortest paths to forward packets [58,62,63]. As a result, routers and links along the shortest path between two nodes may become congested while routers and links along a longer path are idle. The Equal-Cost Multi-Path (ECMP) option of OSPF is useful in distributing load to several shortest paths. But, if there is only one shortest path, ECMP will be useless. So, we need use QoS routing and Constraint Based Routing to solve the problem [59].

A) QoS Routing

QoS Routing refers to algorithms that compute paths that satisfy a set of end-to-end QoS requirements. Given the QoS request of a flow or an aggregation of flows, QoS Routing returns the route that is most likely to be able to meet the QoS requirements [60].

B) Constraint Based Routing

Constraint Based Routing can be used to compute the routes subject to QoS and policy constraints. The goal is to meet the QoS requirements of traffic and to improve utilization of the networks. Constraint Based Routing evolves from QoS Routing, it extends QoS Routing by considering other constraints of the network such as policy, it is used to compute routes that are subject to multiple constraints [61].

Constraint Based Routing is to select optimal routes which most likely meets the QoS requirements of the flows. Using Constraint Based Routing we can select routes to meet certain QoS requirement and increase the utilization of the network. While determining a route, Constraint Based Routing considers not only topology of the network, but also the requirement of the flow, the resource availability of the links, and possibly other policies specified by the network administrators. Therefore, Constraint Based Routing can find a longer and light load path rather than the heavy load shortest path. Network traffic is thus distributed more evenly. In order to do Constraint Based Routing, routers need to distribute new link state information and to compute routes based on such information.

A router needs topology information and resource availability information in order to compute QoS routes. Here, resource availability information means link available bandwidth. Buffer space is assumed to be sufficient and is not explicitly considered. One approach to distribute bandwidth information is to extend the link state advertisements of protocols such as OSPF and IS-IS [58,62,63]. Because link residual bandwidth is frequently changing, a trade-off must be made between the need for accurate information and the need to avoid frequent flooding of link state advertisements. To reduce the frequency of link state advertisements, one possible way is to distribute them only when there are topology changes, or significant bandwidth changes.

The routing table computation algorithms in Constraint Based Routing and the complexity of such algorithms depend on the metrics chosen for the routes. Common route metrics in Constraint Based Routing are monetary cost, bandwidth, reliability, delay, and jitter. Routing algorithms select routes that optimize one or more of these metrics. Metrics can be divided into three classes. Let d(i,j) be a metric for link (i,j). For any path P = (i, j, k, ...,l, m), metric d is:

additive if $d(P) = d(i,j) + d(j,k) + \dots + d(l,m)$
multiplicative if $d(P) = d(i,j) * d(j,k) * \dots * d(l,m)$
concave if $d(P) = \min\{d(i,j), d(j,k), \dots, d(l,m)\}$

According to this definition, metrics delay, jitter, cost are additive, reliability is multiplicative, and bandwidth is concave. Algorithms for finding routes with bandwidth constraint are simple. Bellman-Ford's (BF) Algorithm or Dijkstra's Algorithm can be used [64,65]. For example, to find the shortest path between two nodes with bandwidth greater than 1 Mbps, all the links with residual bandwidth less than 1 Mbps can be pruned first. BF Algorithm or Dijstra's Algorithm can then be used to compute the shortest path in the pruned network. The complexity of such algorithms is $O(N*E)$.

Bandwidth is the useful constraint than delay and jitter, because:

1) Although applications may care about delay and jitter bounds, few applications cannot tolerate occasional violation of such constraints. Therefore, there is no obvious need for routing flows with delay and jitter constraints. Besides, since delay and jitter parameters of a flow can be determined by the allocated bandwidth of the route, delay and jitter constraints can be mapped to bandwidth constraint, if needed.

2) Many real-time applications will require a certain amount of bandwidth. The bandwidth metric is therefore useful.

Some approaches to reduce the computation overhead of Constraint Based Routing include:

1. using a large-valued timer to reduce the computation frequency;
2. choosing bandwidth as constraint;
3. using administrative policy to prune unsuitable links before computing the routing table.

A Constraint Based Routing scheme can choose one of the followings approaches to select a route for a destination.

1. The widest-shortest path, if there are multiple such paths, the one with largest available bandwidth;
2. The shortest-widest path, i.e., a path with largest available bandwidth;
3. The shortest-distance path.

Using paths other than the shortest paths consumes more resources. This is not efficient when the load of the network is heavy. A tradeoff must be made between resource conservation and load balancing. The first approach above is basically the same as today's Dynamic Routing [66]. It emphasizes preserving network resources by choosing the shortest paths. The second approach emphasizes load balancing by choosing the widest paths. The third approach makes a tradeoff between the two extremes. It favors shortest paths when network load is heavy and favors widest paths when network load is mediate [66].

4.4.2 Bandwidth Control

The bandwidth control has been addressed in Chapter 3. The main idea of bandwidth control is to increase the utilization of bandwidth by some means, such as bandwidth allocation, bandwidth sharing, bandwidth borrowing, bandwidth pricing [76,78].

We propose a bandwidth control method, the "Adaptive bandwidth allocation based on dynamic pricing", which we believe it may possibly increase the utilization of bandwidth. The bandwidth price is not fixed, i.e., users can decide to buy bandwidth based on bandwidth price policy. When the price is low, the user can buy more, if all bandwidth requests are over-committed than the available bandwidth, the provider can rise the price to renegotiate, and the users will decrease their requests if the price is high. If the requested bandwidth is equal or almost equal to the available bandwidth, the negotiation is done, the bandwidth is allocated to users. This approach will be presented in Chapter 5 as method to improve the two-bit based DiffServ.

4.4.3 Queue Control

Queue control includes some queue management algorithms, such as RED, RIO, WFQ, WRR, FIFO, etc. These algorithms can support the traffic control to avoid congestion. They can drop the packets to control the length of the queue based on the requirement [75].

4.5 The relationship of DiffServ, Coordinated Control and Price

DiffServ is a kind of QoS model. Coordinated Control is the way to support the DiffServ. There is a very important element for DiffServ—price. We can get different QoS by adjusting the price [71,74]. When the price is higher, the QoS is better. Their relationships are showed in Figure 4.3.

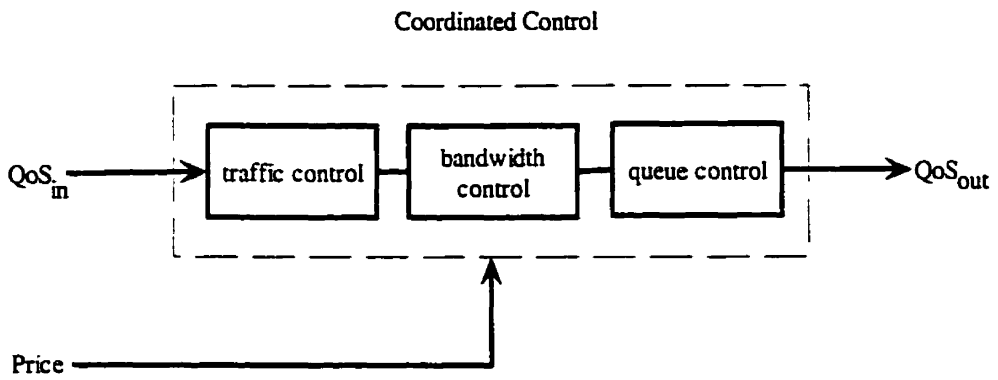
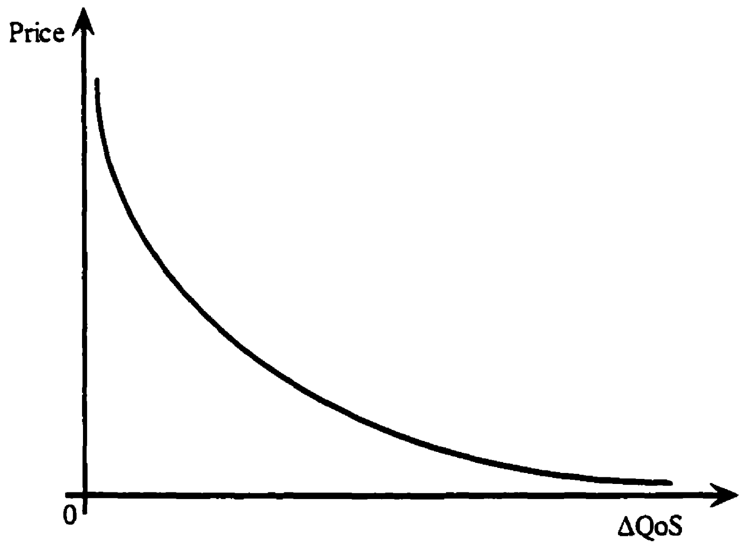


Figure 4.3 A



ΔQoS = QoS in – QoS out

Figure 4.3 B

Figure 4.3 The relationship of QoS, Coordinated Control and Price

Chapter 5 Improving Two-bit Based DiffServ

The DiffServ based on Two-bit (Premium/Assured) architecture has some drawbacks. In this chapter, we focus on the shortcomings of Assured Service, and improve on Assured Service with new algorithms. The simulation code of the algorithms is written in Java and is included in the Appendix.

5.1 Drawbacks of Two-bit based DiffServ

Premium Service and Assured Service are two components of the two-bit DiffServ architecture. Their drawbacks are presented in this section.

A) Premium Service

Premium Service provides low-delay and low-jitter service for customers that generate fixed peak bit-rate traffic. Each customer will have a SLA with its ISP. The SLA specifies a desired peak bit-rate for a specific flow or an aggregation of flows. The customer is responsible for not exceeding the peak rate. Otherwise, excess traffic will be dropped. The Premium Service must keep enough bandwidth based on the peak bit-rate to set up the virtual path. Obviously, the drawback of Premium Service is that it wastes a lot of bandwidth, so that we cannot get high network resource utilization [3,56].

B) Assured Service

Assured Service is intended for customers who need reliable services from their service providers, even during times of network congestion. Customers will have SLAs with their ISPs. The SLAs will specify the amount of bandwidth allocated for the customers. Customers are responsible for deciding how their applications share that amount of bandwidth [3,51].

There are two major drawbacks of Assured Service:

1. Lack of the scalability of service quality;
2. Lack of high utilization of network resources.

We focus on these two shortcomings and propose new methods to overcome them.

5.2 The Scalability of Service Quality in Assured Service

The ratio of data according to type of service in two-bit based Internet traffic in our proposal is as follows: 5% Premium Service traffic, 35% Assured Service traffic, and 60% for Best-effort Service traffic [49]. Because the Premium Service is so expensive, only 5% of traffic packets are premium packets. Most of the DiffServ packets are assured packets, and they represent about 35% of all Internet traffic packets. But, in Assured Service level, all traffic packets are treated equally, even though some customers prefer to pay more for their packets in order to get higher service quality than others. Unfortunately, this cannot be implemented within the current Assured Service level. So, we need to improve the Assured Service model. The current Assured Service level is too coarse, we use multi-class service levels to replace one assured service level, which is called Multilevel Assured Service.

5.2.1 Multilevel Assured Service

The main idea of Multilevel Assured Service is as follows:

A customer pays a minimum basic service charge when he uses Multilevel Assured Service. If he is willing to pay a higher price for a higher level of service, his packets will get a higher level of priority than packets without extra payment. Paying different levels of money for different levels of service will help set different priorities for packets and cause them to enter different priority queues.

High-priority packets are sent earlier than low-priority packets, and they are dropped later than the low-priority packets when there is congestion in the network. We use priority queues to model this approach.

5.2.2 Scheme of Multilevel Assured Service

Multilevel Assured Service has several service levels. The number of service levels is flexible, and may be decided by the service provider based on each particular traffic situation. We consider four service levels, called A1, A2, A3 and A4, with increasing priority and service quality. In other words, A4 has the highest priority and A1 has the lowest. In this scheme, A1 Queue is an assured queue, which is managed by the RIO algorithm. A2 Queue, A3 Queue, and A4 Queue are priority queues with increasing priority. Each queue of A2, A3, A4 is managed by the FIFO algorithm. All these four queues are managed by WFQ algorithm in order to avoid bandwidth starvation. The scheme is shown in Figure 5.1.

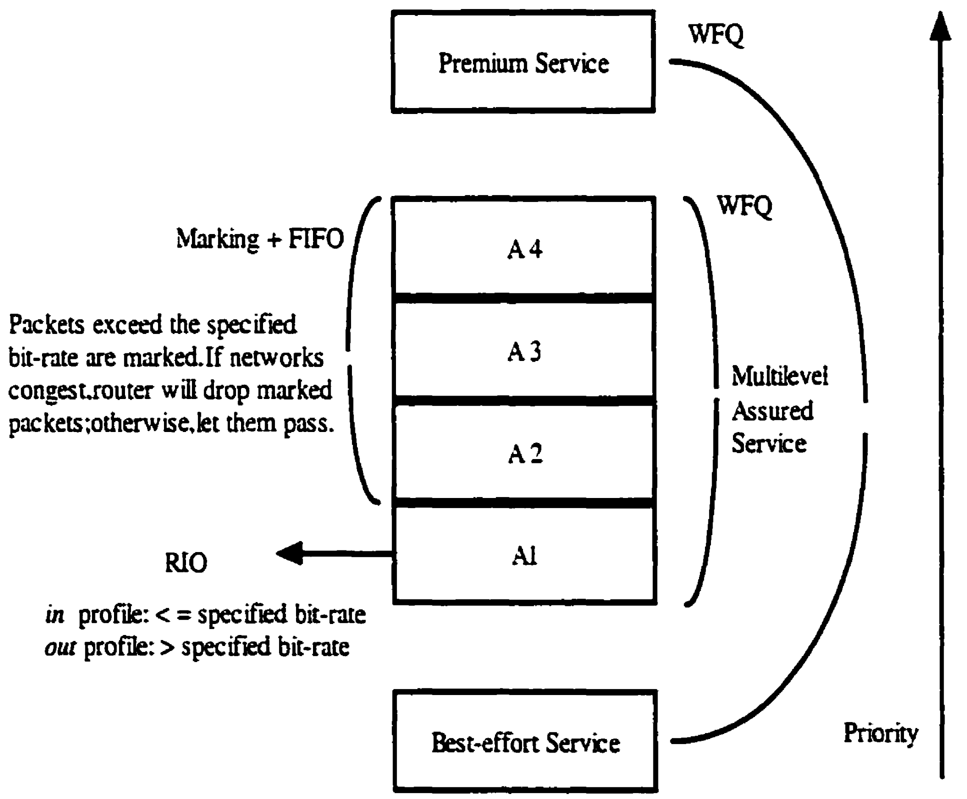

Premium Service
WFQ
Marking + FIFO
A4
A3
A2
A1
WFQ
Packets exceed the specified bit-rate are marked.If networks congest,router will drop marked packets;otherwise,let them pass.
Multilevel Assured Service
RIO
in profile: < = specified bit-rate
out profile: > specified bit-rate
Best-effort Service
Priority

Figure 5.1 The scheme of Multilevel Assured Service

5.2.3 Multilevel Assured Service Supports Scalable Service Quality

Multilevel Assured Service supports scalable service quality by means of Assured Queue and Priority Queue. The Assured Queue is desired for basic assured service, the Priority Queue is destined for high quality service. In our scheme, there is one Assured Queue—A1, and three Priority Queues—A2, A3, A4. The Assured Queue and Priority Queue have been designed for different quality services.

1) Assured Queue

First, if the assured service traffic does not exceed the specified bit-rate, they meet the requirements of the *in* profile. Otherwise, the excess packets are considered as *out* of profile. Second, all packets, *in* and *out*, are put into A1 Queue to avoid out of order delivery. Third, A1 Queue is managed by RIO, which maintains two RED algorithms, one for *in* packets and one for *out* packets. There are two thresholds for each queue. When the queue size is below the first threshold, no packets are dropped. When the queue size is between the two thresholds, only *out* packets are randomly dropped. When the queue size exceeds the second threshold, indicating possible network congestion, both *in* and *out* packets are randomly dropped, but *out* packets are dropped more aggressively. In addition to breaking the TCP flow-control synchronization, RIO prevents, to some extent, greedy flows from hurting the performance of other flows by dropping the *out* packets more aggressively. Because *in* packets have a low loss rate even in the case of congestion, the customers will perceive a predictable service from the network if they keep traffic conformant. When there is no congestion, *out* packets will also be delivered [3].

2) Priority Queue

Priority Queue is managed by FIFO (First In First Out) algorithm and Marking algorithm. If the priority service traffic does not exceed the specified bit-rate, they are sent directly without marking. Otherwise, the exceeding packets are marked first, and then sent. If there is no congestion in the network, both marked and unmarked packets can pass; if congestion happens, unmarked packets can pass but marked packets may be dropped by the router. This is called the "Marking" algorithm [42]. The marking algorithm allows the exceeding bit-rate packets to pass when the traffic load is light, thereby increases the rate of packet delivery and network resource utilization. Without the "Marking" algorithm, all exceeding bit-rate packets are dropped before delivery. Even when the traffic load is very light, they still cannot be delivered, and network resources are wasted. The packets in the Priority Queue are delivered by FIFO algorithm after marking.

Packets in the Priority Queue can receive higher quality service than packets in the Assured Queue, because Priority Queue has a high send-rate and a low drop-rate.

1) High Send-rate
For the high send-rate scheme, Priority packets are sent earlier than assured packets. Assured packets have to wait in the Assured Queue and let packets in the Priority Queue be delivered first. In order to avoid starving the Assured Queue, we can use WFQ to manage the Priority Queue and the Assured Queue.

2) Low drop-rate
For the low drop-rate scheme, marked priority packets will be dropped only after congestion happens. But assured packets will be dropped by the RIO algorithm not only when congestion has become real, but also when the trend towards network congestion has been detected.

5.2.4 Choosing a Fitting Service Level

A customer needs to choose a fitting service level before using the Multilevel Assured Service. We consider there are four service levels in our Multilevel Assured Service scheme, which are A1, A2, A3 and A4 with increasing priority. A1 is for basic assured service, the customer only needs to pay a basic service charge for it, say *X* dollars. If the customer is willing to pay more to get higher priority for his packets, he can choose a priority queue (A2, A3 or A4) to satisfy his requirement based on two elements: the price and the size of each priority queue. The price of each priority queue can be set by the service provider. Suppose the service price for A2 is *2X* dollars, A3 is *3X* dollars, and A4 is *4X* dollars. The size of the queue is given by the length of all existing packets in the queue. Since the priority queue is managed by FIFO algorithm, and if the queue size is too large, the new packets have to wait for the service for a longer time. So, if the packets are important and urgent for the sending application, a higher priority queue will be chosen.

If the customer is satisfied with both the price and the size of the queue, the service level can be set. After setting the service level, the customer's application sends the packets to the network, the network will process the packets based on the setting. The implementation of Multilevel Assured Service is presented in section 5.4.4.

5.3 The Utilization of Network Resources

For current Assured Service, the network resource cannot be utilized efficiently. At the source ISP, it is hard to allocate all available bandwidth completely for the requestors, either when bandwidth is over-committed or under-committed, so the bandwidth utilization is low. Between the source ISP and destination ISP, when the spatial granularity becomes larger than one destination, it is more difficult to support a service with a fixed bandwidth profile [67,68]. The network needs to provide enough resources to all possible destinations to ensure the service quality and thus the resource utilization is low.

We propose to use Token-based Assured Service and Constraint Based Routing in order to address the above problem. In order to improve the bandwidth utilization at the source ISP, we propose a new algorithm, called Token-based Assured Service. The Constraint Based Routing, as we have presented in Chapter 4, can select routes to meet the QoS requirement and increase the resource utilization between the source ISP and the destination ISP.

To ensure high network resource utilization, we propose the following stages: (1) Using Token-based Assured Service to increase the bandwidth utilization at the source ISP, (2) Using Constraint Based Routing to increase the network resource utilization between the source ISP and destination ISP, and (3) Using Load Balancing to support the Constraint Based Routing if necessary.

5.3.1 Token-based Assured Service

There are two major elements of the Token-based Assured Service: (1) tokens, and (2) token price. Tokens stand for bandwidth amount. For example, if a network has 100M bandwidth and every token stands for 1M, then the network has 100 tokens in total. If a user needs 5M bandwidth, he should buy 5 tokens. Token price stands for bandwidth price. That is to say, $10 for one token means $10 for 1M bandwidth. The available token number is the number of tokens available for allocating at the mean time. In another words, it is the unused bandwidth in the network.

Tokens can be used to avoid over-committing the bandwidth. All packets must hold some tokens before entering the network, which means the bandwidth must be allocated to the packet first. No token means no bandwidth, so there is no entrance for any packet. A packet must hold more tokens than its minimum token requirements; otherwise, the packet cannot be delivered properly.

The service provider allocates tokens to all requestors based on the dynamic token price, until all tokens are allocated completely.

5.3.2 Dynamic Token Price

Dynamic token price is the tool used to allocate tokens. When the number of available tokens is greater than the number of total requested tokens, the bandwidth is under-committed. In this case, the service provider needs to reduce token price to encourage the requestors to buy more tokens. When the token price is reduced, the requestors will buy more tokens. Because more tokens means more bandwidth, and it is easier to get better QoS, the token price will be reduced until all or almost all tokens have been sold.

If the available tokens are fewer than total requested tokens, the service provider will increase the token price thus trying to reduce requestor demand for tokens. When the token price is increased, the number of tokens requested will be decrease. Normally, the customer will request a greater number of tokens than his minimum token requirement.

Having a dynamic token price can avoid abuse of the bandwidth. Because the bandwidth is not free, and every one has to pay for it based on the amount used. So, every one needs to apply for a reasonable amount of bandwidth.

It is very hard to give a reasonable fixed bandwidth price, because if the price is too low, some greedy users will abuse it; if it is too high, it will prohibit increased use. So, we use dynamic token pricing, which is fair for every user. A user can buy the exact number of tokens he needs. When there are fewer tokens available, the token price will be high. In that case, a user can wait until the lower token price is available, or buy tokens with a higher price for delivering his urgent packets.

Dynamic token price provides a chance for some users who want to get high priority. When total requested tokens are much more than the available tokens, a user can bid for tokens against others for his important packets. This means users can pay more to get priority for delivering packets instead of waiting.

5.3.3 Token-based Assured Service Supports High Utilization of Bandwidth

As described above, token-based assured service can support high bandwidth utilization in two ways:

1) Avoids over-committing the bandwidth based on token-holding entrance.
2) Dynamically adjusts of the token price to encourage or discourage the users to buy more tokens or to reduce tokens requirements, until all available tokens are sold.

The implementation of Token-based Assured Service is given in the section 5.4.5.

5.3.4 Increasing the Resource Utilization by Constraint Based Routing

After assigning all available tokens to the requestors, the ISP source will be ready to process users' packets. The ISP delivers these packets based on Constraint Based Routing. The Constraint Based Routing algorithm can find a longer and less loaded path rather than a more heavily loaded, shorter path, which increases the network utilization between the source and the destination. Load balancing can reduce the traffic load by choosing the widest path, and it can be used concurrently with Constraint Based Routing.

We consider that using our method, the bandwidth utilization is more efficient at the ISP source than before. From ISP source to ISP destination, when the packets pass through the Internet backbone, Constraint Based Routing and load balancing can ensure high resource utilization, hence the whole network resource utilization is higher than before.

5.4 Implementation

We propose two implementations, one is for Multilevel Assured Service, the second is for Token-based Assured Service. Both of them are based on Agent-Broker architecture, see Figure 5.2.

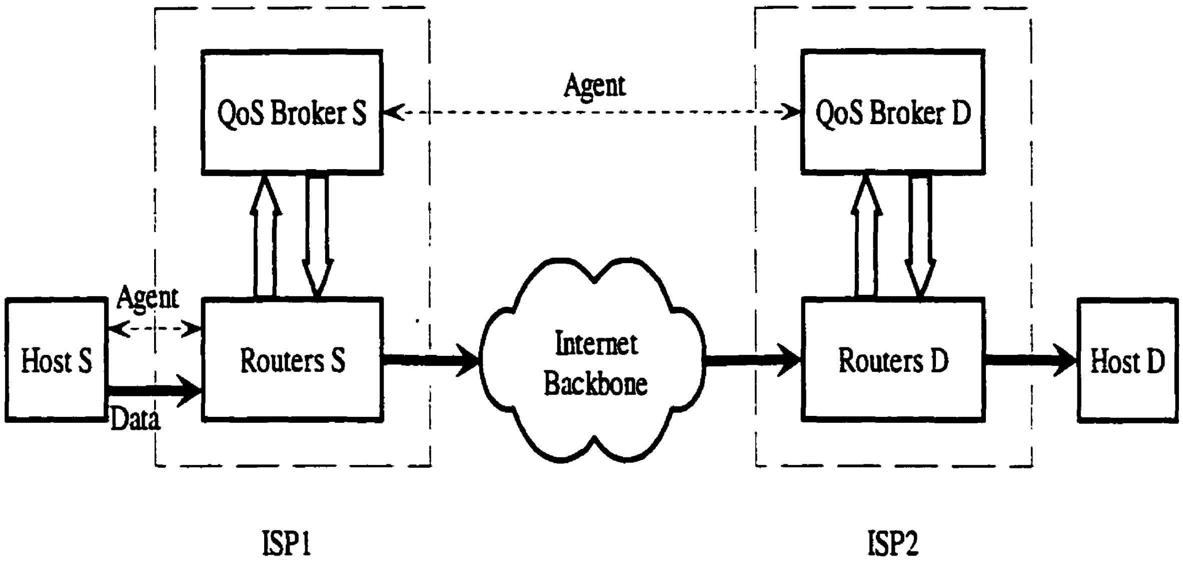


Figure 5.2 Agent-Broker Architecture

5.4.1 Agent-Broker Architecture

Agent-Broker architecture can support the implementations of Multilevel Assured Service and Token-based Assured Service. The Agent-Broker architecture is shown in Figure 5.2.

In Figure 5.2, S stands for Source, D stands for Destination. So, Host S means Source Host. QoS Broker S means Source QoS Broker. Routers S means Source routers, including boundary routers, and core routers, etc. Similarly, Host D means Destination Host, QoS Broker D means Destination QoS Broker, and Routers D means all Destination routers.

In Agent-Broker architecture, the Host is a service requestor, the ISP is a service provider. The Host sends an agent to the ISP to negotiate with the QoS Broker on its behalf. The Agent, as we described in 3.6.2, is a small program that can run on remote machines. The Agent includes some QoS parameters, such as service level requirement, bandwidth requirement, bit-rate, etc. The ISP includes one QoS Broker and some routers. The QoS Broker can negotiate with agents, assign system resources to them, and satisfy their QoS requirements, such as service level, bandwidth amount, etc. The QoS Broker can also communicate with other remote QoS Brokers. Routers within the ISP, and they can be classified as either boundary routers or core routers. Boundary routers are connected to hosts and control classification, marking, policing, shaping operations, etc. Core routers control the forwarding of packets, and they are connected to the Internet backbone.

In Figure 5.2, if Host S wants to transmit data to Host D, it will send an Agent to Routers S first. The boundary router of Routers S receives the Agent and runs it there. The boundary router invokes the QoS Broker S based on the Agent's QoS requirement. The Broker S responds to the boundary router, and then the Agent negotiates with the Broker S. If the Agent's requirement is accepted, the Broker S will forward the Agent to QoS Broker D; otherwise, an error message will be sent back to Host S by the Agent. When QoS Broker D receives the Agent, it will negotiate with the Agent. If the requirement is accepted, the Broker D will set the classification and policing rules on Routers D, and send the Agent back to Broker S with a confirmed message; otherwise, it will only send an error message to Broker S by the Agent. When the Broker S receives a confirmed message from Broker D, it will set the classification and policing rules on Routers S, and then send the Agent back to Host S with a confirmed message; otherwise it will just send back the Agent with an error message. When the Host S receives a confirmed message from Broker S, it can start transmitting data. The data will pass through the network from Routers S, through the Internet backbone, and Routers D to Host D. If the Host S receives an error message from Broker S, Host S may change its QoS requirement, and then sends a new Agent to renegotiate with the Broker S.

5.4.2 Agent-Broker Negotiation

Normally, an ISP has a lot of hosts, and all the hosts can send agents to negotiate with the Broker for service on their behalf. An example of four Hosts sending four Agents to the ISP is shown in Figure 5.3.

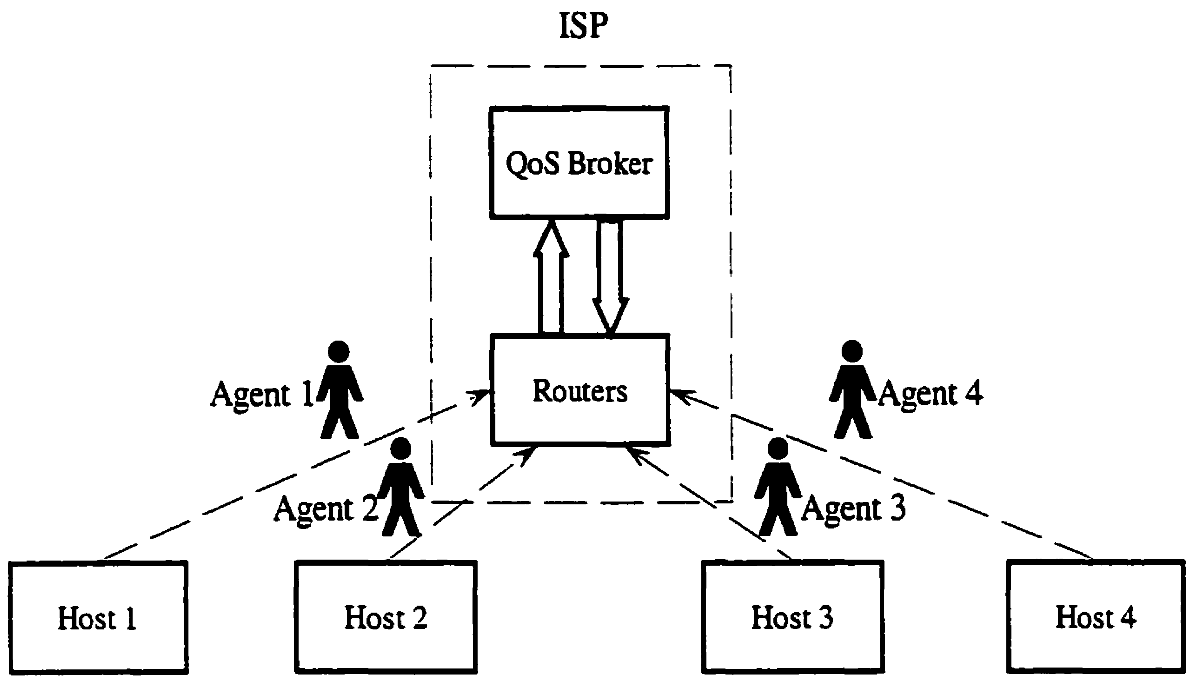


Figure 5.3 Hosts send agents to ISP

As presented in Figure 5.3, Routers includes boundary routers and core routers. The boundary router of the ISP receives the four Agents, and it invokes the QoS Broker to negotiate with the Agents. All four Agents can negotiate with the Broker simultaneously. The negotiation of the Agent and the QoS Broker is shown in Figure 5.4.

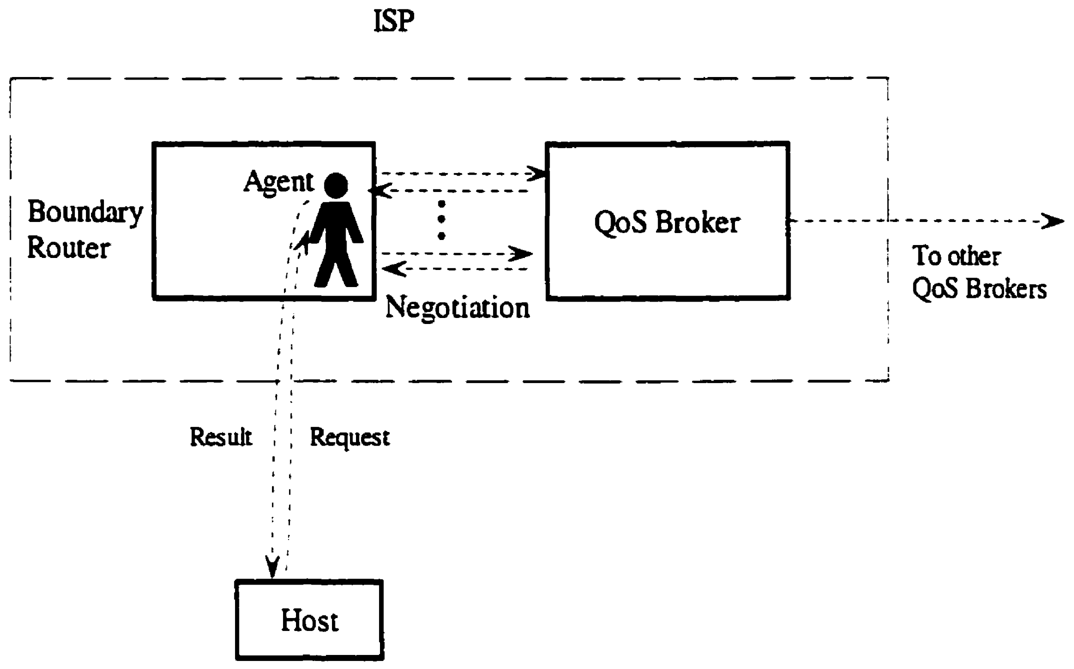
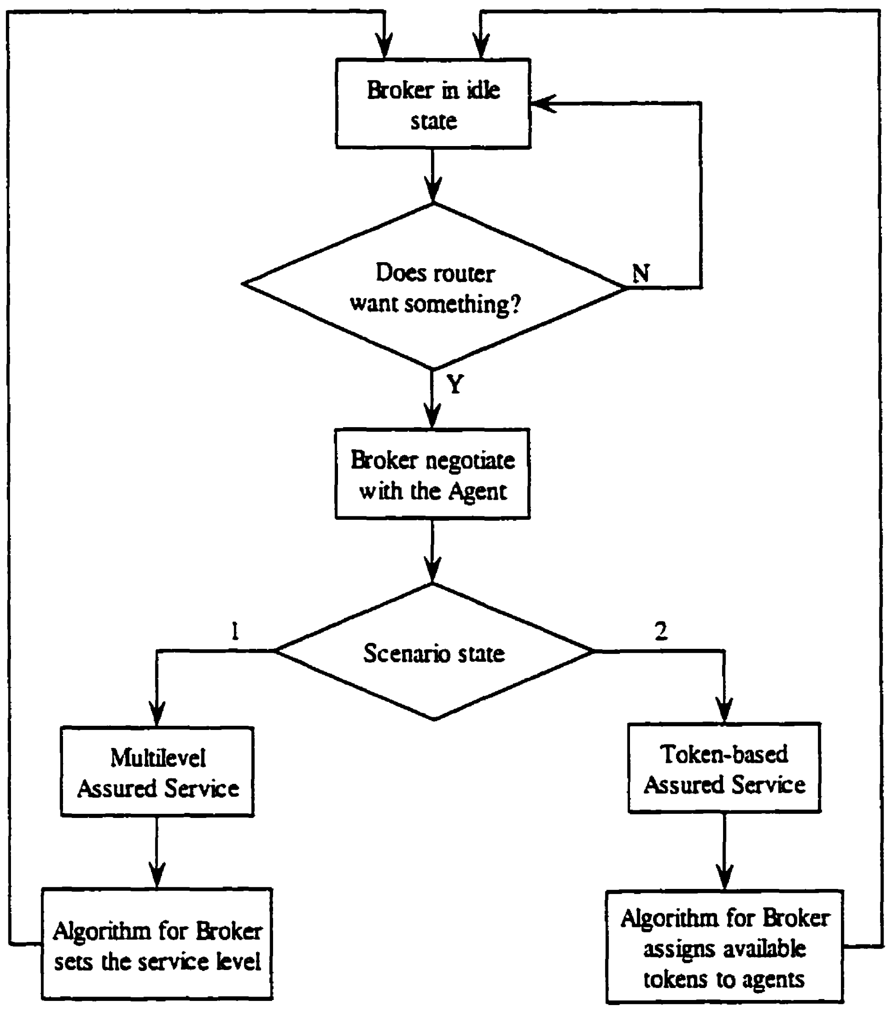


Figure 5.4 The negotiation of Agent and QoS Broker

In Figure 5.4, the Host requests service by sending an Agent to its ISP. The Agent is installed in the Boundary Router to negotiate with the QoS Broker there. The negotiation may last several rounds, until Agent and Broker get a mutually satisfying result, which will be taken back to the Host by the Agent. If the Agent's requirement cannot be satisfied, the Agent will return to the Host with an error message.

5.4.3 Broker Behavior

The Broker behavior in our implementation is shown in Figure 5.5.



Broker in idle state
Does router want something?
N
Y
Broker negotiate with the Agent
Scenario state
1
2
Multilevel Assured Service
Token-based Assured Service
Algorithm for Broker sets the service level
Algorithm for Broker assigns available tokens to agents

Figure 5.5 QoS Broker behavior

The Broker is in an idle state at first, while waiting for the request from the router. When the router receives Agents, it invokes the Broker. The Broker will negotiate with Agents how to set the service level and how to assign all available tokens.

5.4.4 Implementation of Multilevel Assured Service

The implementation of Multilevel Assured Service is based on Agent-Broker architecture, as shown in Figure 5.2.

The Host sends an Agent to ISP on its behalf to negotiate the service level, as shown in Figure 5.3. There are many Hosts, all of them can send Agents to the Broker to request service. Here, we give an example of one Host sending one Agent to negotiate the service level with the Broker, as presented in Figure 5.4.

A) Service Level Establishment Scenario

Multilevel Assured Service includes four queues, A1, A2, A3 and A4. A1 queue charges a basic price, while A2 charges double, A3 triple and A4 four times as much as A1. Hosts send out their Agents to negotiate with the QoS Broker. First, at the Host side, the Agent gets information about the maximum service price and the maximum queue size that the Host can accept. Second, the Agent is sent to the ISP. Third, after the boundary router receives the Agent, the negotiation of Agent and Broker starts.

The Agent gets the price of the A1 level, and if the price is higher than the Agent's maximum price, the negotiation process is terminated and Best-effort Service is allocated to the Agent. Otherwise, the agent gets the queue size of A1, and if the size is acceptable, A1 is assigned to the Agent. But if the queue size of A1 is too large, the agent has to give up on A1, and asks for the price of A2. Likewise, if the price and queue size of A2 are both acceptable, A2 Queue service can be assigned to the Agent. If unfortunately, the price is too high, the Agent is assigned to A1, and if the size of A2 is too large, the Agent will go on to seek A3 queue. Then the process will continue with A3 and A4 queue with the same strategy.

B) Service Level Establishment Algorithm

The service level establishment algorithm can be divided into two parts. One is the Broker which sets the service level, the other is the Agent which asks for the service level.

1. The algorithm for Broker sets service level is shown in Figure 5.6.

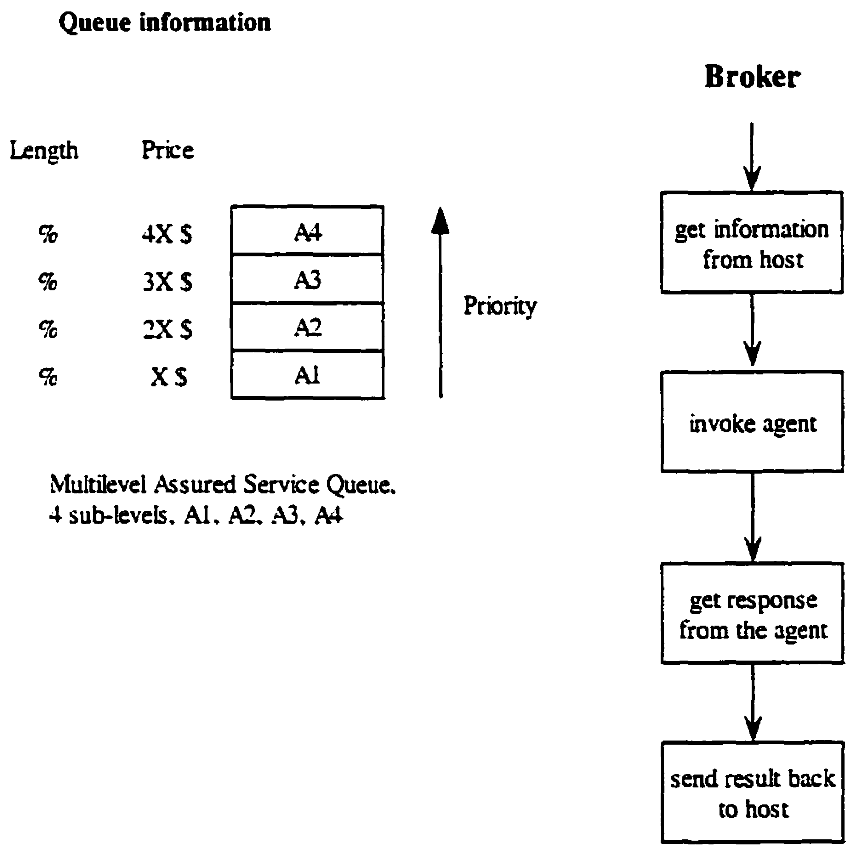

Queue information
Length
Price
%
4X $
3X $
2X $
X $
A4
A3
A2
A1
Priority
Multilevel Assured Service Queue.
4 sub-levels. A1. A2. A3. A4
Broker
get information from host
invoke agent
get response from the agent
send result back to host

Figure 5.6 Broker sets the service level

2. The algorithm for the Agent asks for the service level is shown in Figure 5.7.

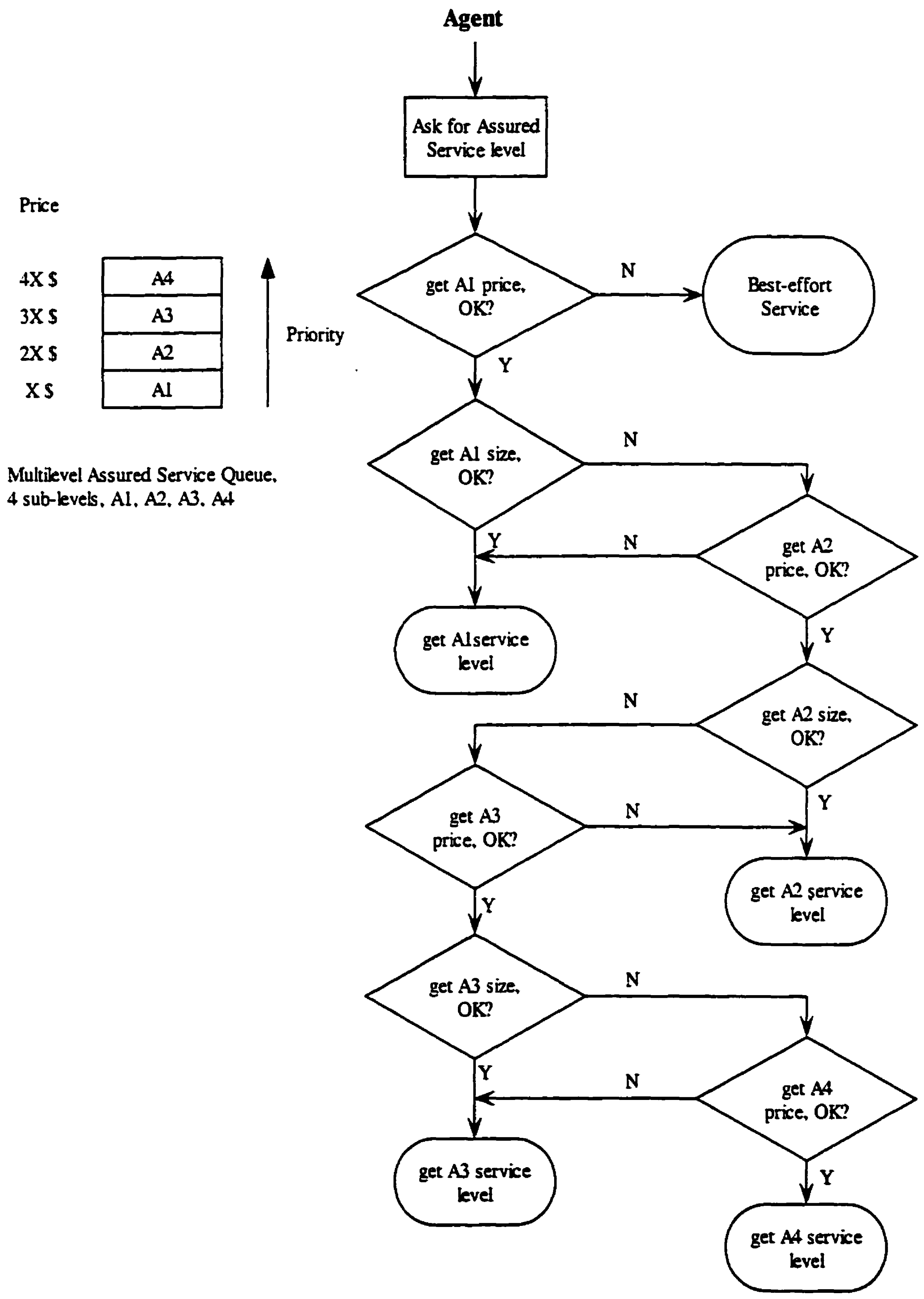


Figure 5.7 Agent asks for service level

The service level can be set by the above two algorithms. After the service level is established, the result is sent back to the Host. When the Host receives the result, it sends the packets to the ISP. The ISP puts the packets in the queue. The packets will be delivered by the router later.

5.4.5 Implementation of Token-based Assured Service

The implementation of Token-based Assured Service is based on this Agent-Broker architecture as well, as presented in Figure 5.2.

The hosts send Agents to the ISP on their behalf to negotiate with the Broker for tokens (bandwidth amount).

The hosts must hold some tokens before entering the network, and these tokens should be at least somewhat more than their minimum bandwidth requests. All hosts are eager to get more tokens, because more tokens means more bandwidth, so that the QoS will be easy to meet. But, the bandwidth is always limited, and it is not free. So, the requestor will submit an initial request (preferred tokens) which is a bit more than its minimum request, and then modify the request according to the change of the token price. The token price is dynamic, because it can both avoid the greedy requestors and invoke the idle requestors.

If there are more token requested than the available tokens, the token price will rise until the requested and available tokens are balanced (equilibrium point is reached). If the requested tokens are less than the available tokens, the token price will fall until all available tokens are sold out or almost sold out.

A) Scenario of Negotiating Tokens

Four hosts (Host 1, Host 2, Host 3, Host 4) send four Agents (Agent 1, Agent 2, Agent 3, Agent 4) to the ISP to ask for their preferred tokens. The preferred tokens are more than their minimum requested tokens. The ISP's boundary router receives the requests from four Agents, and invokes the Broker to negotiate with the Agents.

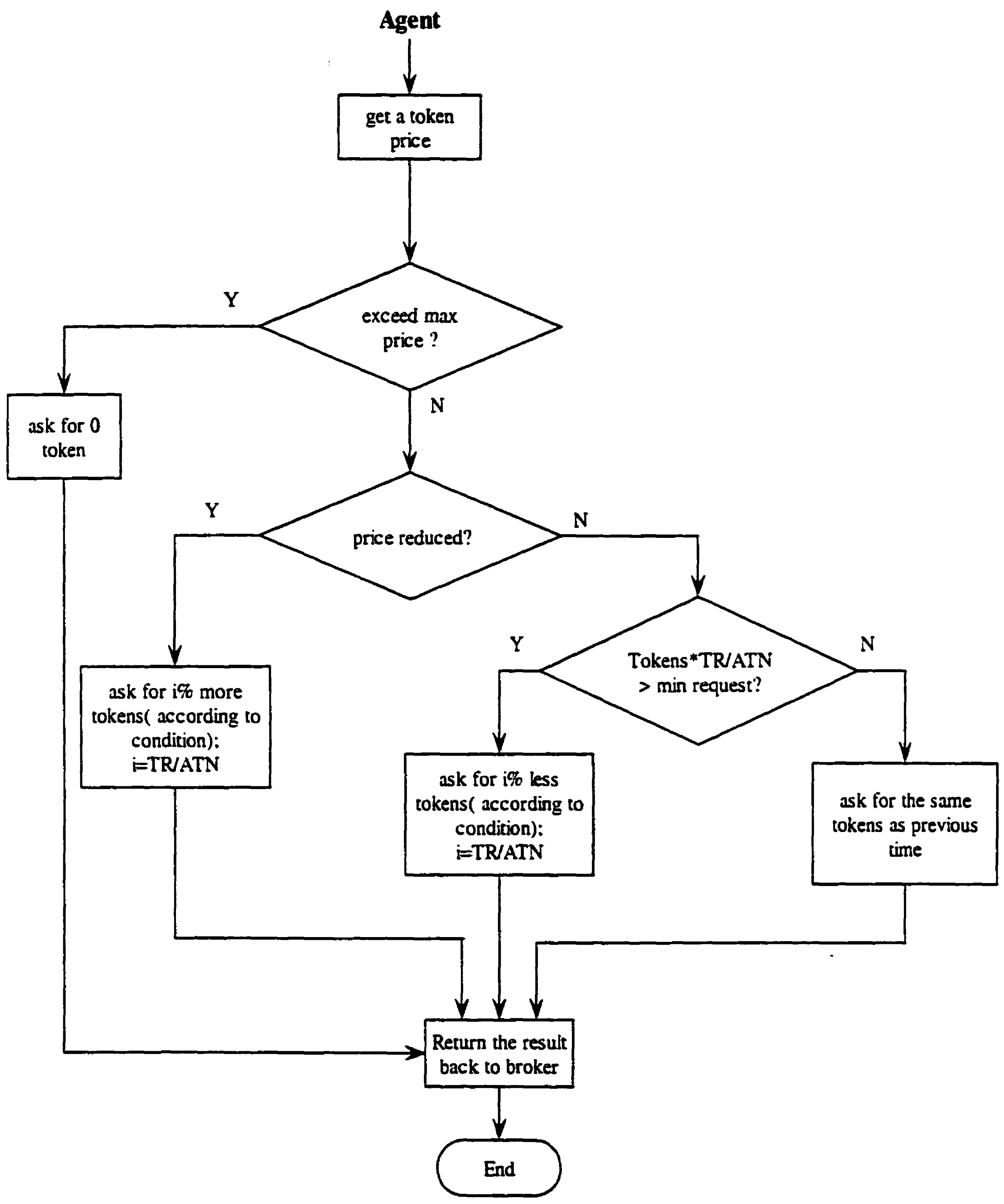
The Broker states an initial token price and collects all requests of the Agents; then it compares the number of total requested tokens with the number of available tokens, and states a new token price. If the total available tokens are much more than the requested tokens, the token price is lowered; otherwise, it is increased.

After the Broker states a new token price, all Agents can get the new price information, and compare it with the previous token price. The Agent will increase its request if price is lowered, or vice versa. The Broker then collects requests from Agents again. If new requested tokens are still fewer than the available tokens, the Broker reduces the token price again, encouraging the agents to buy more tokens, until all available tokens are sold. If the new requested tokens are more than the available tokens, the Broker rises the token price. If the token price is higher than the Host's maximum acceptable price, the Agent has to quit (asks for zero token); otherwise, the Agent reduces its token request as long as the request does not go lower than its minimum request. Then the Agent applies for tokens again. If the request is accepted by the Broker, the Agent returns to its Host with the result; if not, it renegotiates. If the Agent's minimum token request cannot be satisfied, the Agent has to quit this round of negotiation. This Agent can wait for the next round of negotiation, when it can probably get low-priced tokens.

B) Algorithms for negotiating tokens

There are two algorithms for negotiating tokens: (1) Agent asks for tokens, (2) Broker assigns tokens.

1. The algorithm "Agent asks for tokens" is shown in Figure 5.8.


Agent
get a token price
exceed max price ?
Y
N
ask for 0 token
price reduced?
Y
N
ask for i% more tokens(according to condition): i=TR/ATN
Tokens*TR/ATN > min request?
Y
N
ask for i% less tokens(according to condition): i=TR/ATN
ask for the same tokens as previous time
Return the result back to broker
End

TR: total request
ATN: available token numbers

Figure 5.8 Agent asks for tokens (bandwidth amount)

The key idea of the algorithm of the Agent asking for tokens is described as follows. Based on the dynamic token price, the Agent tries to apply for as many tokens as possible. There are two ways to do that:

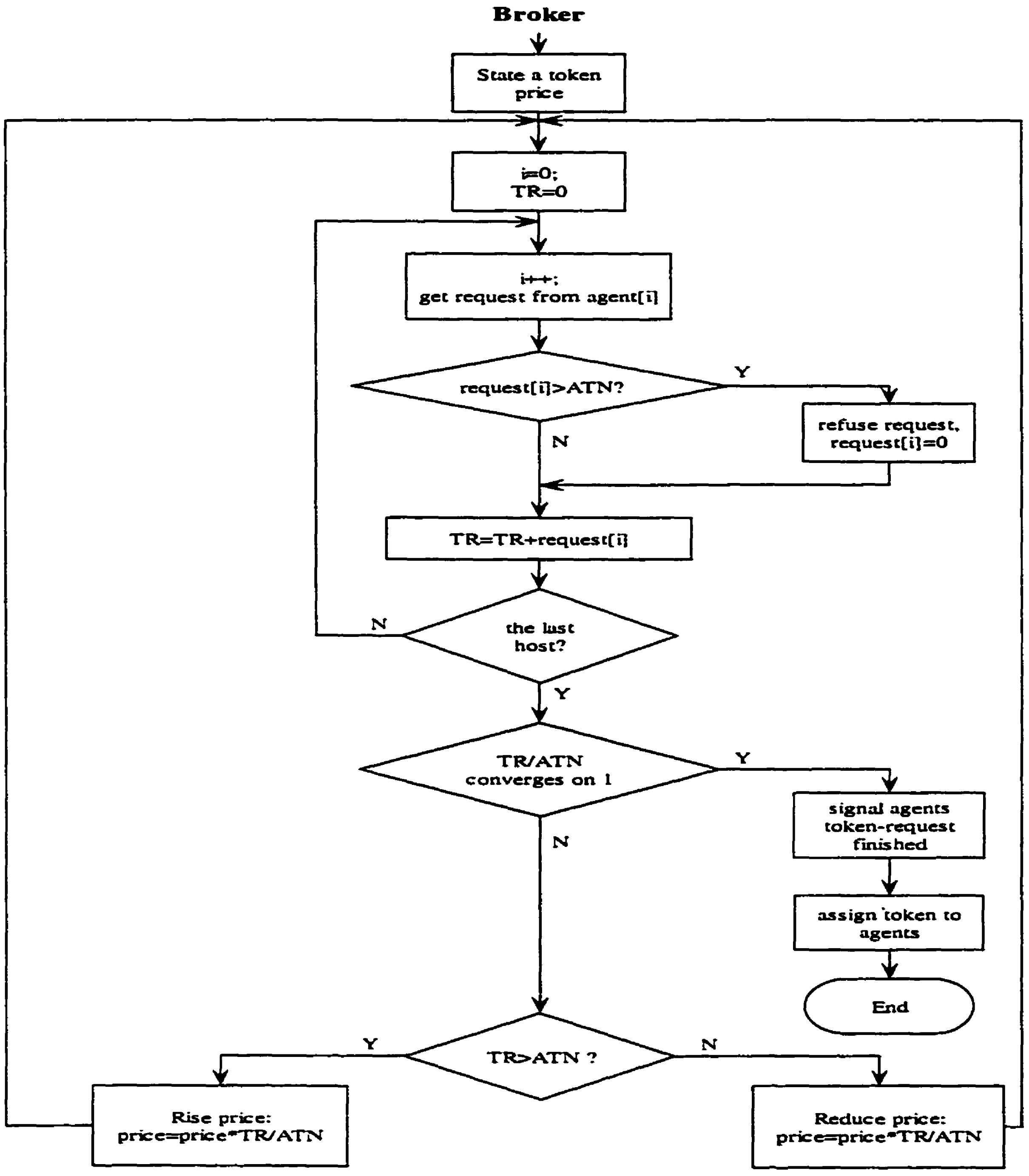
a) When the token price is going down, increase the token requests.
b) When the token price is going up, reduce the token request, until it reaches the minimum token request. In that case, the Agent can give up this round of negotiation, and wait for next round to get the appropriate token price and number of tokens.

2. The algorithm "Broker assigns tokens" is shown in Figure 5.9.

The key idea of the algorithm of the Broker assigning all available tokens to Agents is as follows.

The Broker tries to sell all its available tokens by dynamic token pricing, which means that all available bandwidth will be used, so that the highest possible rate of bandwidth utilization is achieved.

After the Broker assigns the available tokens to all Agents, the Broker signals the router for service. The Agents send back the final negotiation results to the Hosts, and the Hosts can send their packets to the ISP.


Broker
State a token price
i=0; TR=0
i++; get request from agent[i]
request[i]>ATN?
Y
refuse request, request[i]=0
N
TR=TR+request[i]
the last host?
N
Y
TR/ATN converges on 1
Y
signal agents token-request finished
assign token to agents
End
N
TR>ATN ?
Y
N
Rise price: price=price*TR/ATN
Reduce price: price=price*TR/ATN

TR: total request
ATN: available token numbers

Figure 5.9 Broker assigns all available tokens to agents

5.5 Simulation

The simulation program has been written in Java.

The Agent is an application program, which can be sent from local host to the server (ISP) by FTP protocol. The simulation source code is given in the Appendix. The source code includes two parts, part one is for service level negotiation; part two is for bandwidth negotiation.

Chapter 6 Conclusion and Future Work

The Internet is used on a very large scale nowadays. QoS introduction in the Internet becomes necessary to support different service requirements. DiffServ is the first step of QoS implementation in the Internet.

We have focused on the two-bit DiffServ architecture in the Internet, because two-bit architecture has the outstanding advantage of being easy to implement. In the foreseeable future, it will possibly be used widely in the Internet. The current two-bit architecture has some drawbacks. We present some ways to improve them, such as Multilevel Assured Service, the Marking algorithm, the Token-based Assured Service, the Agent-Broker Algorithm, Constraint-based Routing and load balancing, etc. Using these methods, we can get a higher scalable service quality and higher network resource utilization. These improvements can make the two-bit architecture more efficient than it is right now. We believe that with our improvements, the two-bit DiffServ architecture will have a stronger capacity to support different QoS requirements.

Future research should be conducted to extend our work in order to support multiple ISP environments, multicast communications, and both sender and receiver-based charging schemes in the two-bit DiffServ architecture.

References

1. A. Tanenbaum, *Computer Networks*, Third edition.
2. P. Ferguson and G. Huston, *Quality of Service*, John Wiley & Sons, 1998
3. K. Nichols, V. Jacobson and L. Zhang, *A Two-bit Differentiated Services Architecture for the Internet*, Internet Draft, Nov. 1999.
4. J.J. Bae and T.Suda, *Survey of Traffic Control Protocols in ATM Networks*, December 1990
5. Kevin Lai, Mary Baker, *Measuring Bandwidth*, Dept. of computer Science, Stanford University.
6. T. Ndousse and L. Hester, *PPP Extensions for IP/PPP-HDLC over SONET-SDH/ WDM*, Proceedings of the 16th International Conference on Communications, June, 1999
7. A. Campbell, *A Quality of Service Architecture*, Ph.D. thesis, January 1996.
8. ISO, *QoS-Methods and Mechanism, International Standards Organization*, 1998
9. V. Jacobson, *Congestion Avoidance and Control*, ACM, 1988
10. Timothy Kwok, *ATM: The new paradigm for Internet, Intranet, and Residential Broadband Services and Applications*, Prentice Hall PTR, New Jersey 07458
11. Zbigniew Dziong, *ATM Network Resource Management*, McGraw-Hill
12. Sally Floyd and Kevin fall, *Router Mechanisms to Support End-to-End Congestion Control*, Lawrence Berkeley National Lab, Feb. 15,1997
13. Jeffrey K. MacKie-Mason, Hal R. Varian, *Pricing the Internet*, University of Michigan, February 1994
14. Zygmunt Haas, *Adaptive Admission Congestion Control*, AT&T Bell Lab.
15. Jean Walrand, Pravin Varaiya, *High-Performance Communication Networks*, Morgan Kaufmann Publishers, Inc.
16. Raj Jain, *Congestion Control and Traffic Management in ATM Networks: Recent Advances and A Survey*, Dept. of CIS, The Ohio State University, Aug.13,1996.
17. Dominique Gaiti and Guy Pujolie, *Performance Management issues in ATM Networks: Traffic and Congestion Control*, IEEE/ACM Transaction on Networking,

Vol.4, No.2, April 1996.

18. Jose Duato, Sudhakar Yalamanchili, Lionel Ni, *Interconnection Networks an engineering approach*, IEEE Computer Society, Los Alamitos, California
19. A. Parekh, *A Generialized Processor Sharing Approach to Flow Control in Integrated Services Networks*, Ph.D. thesis, MIT, Feb. 1992
20. K. Ramakrishnan, and R. Jain, *A Binary Feedback Scheme for Congestion Avoidance in Computer Networks*, ACM Transaction on Computer Systems, Vol.8, No.2, 1990
21. Sudhir S. Dixit, *Traffic Descriptor Mapping and Traffic control for Frame Relay Over ATM Network*, IEEE/ACM Transaction on Networking, Vol.6, No.1, Feb. 1998.
22. L. Petersen, B. Davie, *Computer Networks – A Systems Approach*, Morgan Kaufmann Publishers, San Francisco, CA; 1996.
23. S. Radhakrishnan, S.V. Radghavan, a. Agrawala, *Design & Performance Study of a Flexible Traffic Shaper for High Speed Networks*, Feb.1997
24. Edward W. Knightly and Jingyu Qiu, *Measurement-Based Admission control with Aggregate Traffic Envelopes*, ECE Department, Rice University.
25. S. Doran, *RED Experience and Differentiated Queuing*. In NANOG Meeting, June 98
26. S. Floyd, *TCP and Explicit Congestion Notification*, Computer Communication Review, October 1994.
27. H. Liu, *Traffic Shaping for Congestion Control in High Speed ATM Networks*, Dept. of CS, University of Saskatchewan, August 1992.
28. K. Bala, I. Cidon and K. Sohraby, *Congestion control for High Speed Packet Switched Networks*, June 1990.
29. R.Braden, L. Zhang, S. Berson, S. Herzog and S. Jamin, *Resource ReSerVation Protocol (RSVP)* RFC 2205, September 1997.
30. *Cisco Billing Architecture White Paper*, Cisco Systems, Inc. 1998
31. Mohsen Guizani and Ammar Rayes, *Designing ATM Switching Networks*, McGraw -Hill
32. Ivy Hsu and Jean Walrand, *Admission Control for ATM Networks*, Minneapolis, Minnesota, March 1994
33. L. Zhang, *Virtual Clock: A new Traffic Control Algorithm for Packet-Switched Networks*, ACM Transactions on Computer Systems, Vol. 9, No. 2, May 1991

34. Arthur W. Berger, Ward Whitt, *Effective Bandwidth with Priorities*, IEEE/ACM Transaction on Networking, Vol.6, No.4, August 1998.

35. Alfonso Fuggetta, Gian Pietro Picco, Giovanni Vigna, *Understanding Code Mobility*, IEEE Transactions on Software Engineering, Vol 24, 1998

36. Erol Gelenbe, Xiowen Mang, Raif Onvural, *Bandwidth Allocation and call Admission control in High-Speed Networks*, IEEE Communication Magazine, May 1997.

37. *CA*net II Differentiated Services, Bandwidth Broker System Specification*, British Columbia Institute of Technology, Technology Centre, Group for Advanced Information Technology, October, 1998

38. *Mobile Agents White Paper*, http://www.genmagic.com/technology/techwhitepaper. Html, General Magic, Inc., 1997

39. Tony White, Bernard Pagurek, Andrzej Bieszczad, *Network Modeling for Management Applications Using Intelligent Mobile Agents*, Carleton Unversity, 1999

40. Steward Snell, Jtec Pty Limited, *Dynamic Bandwidth Management Using ATM*, 1999

41. D. Tennenhouse, D. Wetherall, *Towards an Active Network Architecture*, April 1996

42. *Cisco Access VPN white paper*, Cisco Systems, Inc. 1998

43. S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, *An Architecture for Differentiated Services*, RFC 2475, Dec. 1998.

44. Braden, R., Clark, D. and Shenker, S., *Integrated Services in the Internet Architecture: an overview*, Internet RFC 1633, Jun. 1994

45. J. Wroclawski, *Specification of the Controlled-Load Network Element Service*, RFC 2211, Sept. 1997

46. T. Li, *CPE based VPNs using MPLS*, Internet draft <draft-li-MPLS-vpn-00.txt>, Oct. 1998

47. R. Braden, L. Zhang, S. Berson, S. Herzog and S. Jamin, *Resource ReSerVation Protocol (RSVP) --Version 1 Functional Specification*, RFC 2205, Sept. 1997

48. P. Vaananen and R. Ravikanth, *Framework for Traffic Management in MPLS Networks*, Internet draft<draft-vaananen-mpls-tm-framework-00.txt>, Mar. 1998

49. Y.Bernet et al., *A Framework for Differentiated Services*, Internet draft <draft-ietf-diffserv-framework-00.txt>, May 1998

50. K. Nichols, S. Blake, F. Baker and D. Black, *Definition of the Differentiated*

Services Field (DS Field) in the IPv4 and IPv6 Headers, RFC 2474, Dec. 1998.

51. K. Nichols et al., *Differentiated Services Operational Model and Definitions,* Internet draft <draft-nichols-dsopdef-00.txt>, Feb. 1998
52. Kimberly C. Claffy, *Internet Traffic Characterization,* University of California, San Diego, 1994
53. Silvano Gai, *Internetworking IPv6 with Cisco Routers,* McGraw-Hill
54. D. Clark and J. Wroclawski, *An Approach to Service Allocation in the Internet,* Internet draft <draft-clark-different-svc-alloc-00.txt>, Jul. 1997
55. B. Braden et al., *Recommendation on Queue Management and Congestion Avoidance in the Internet,* RFC 2309, Apr. 1998
56. V. Jacobson, *Differentiated Services Architecture,* talk in the IntServ WG at the Munich IETF, August, 1997
57. Y.Bernet et al., *A Framework for use of RSVP with DiffServ Networks,* Internet draft <draft-ietf-diffserv-rsvp-00.txt>, Jun. 1998
58. R. Guerin, S. Kamat, A. Orda, T. Przygienda, and D. Williams, *QoS Routing Mechanisms and OSPF extensions,* Internet draft <draft-guerin-QoS-routing-ospf-03.txt>, Jan. 1998
59. Q. Ma, *QoS Routing in the Integrated Services networks,* Ph.D. thesis, CMU-CS-98-138, Jan. 1998
60. E. Crawley, R. Nair, B. Jajagopalan and H. Sandick, *A Framework for QoS-based Routing in the Internet,* RFC 2386, Aug. 1998
61. Z. Wang and J. Crowcroft, *Quality of Service Routing for Supporting Multimedia Applications,* IEEE JSAC, Sept. 1996
62. C. Villamizar and T. Li, *IS-IS Optimized Multipath (IS-IS OMP),* Internet draft <draft-villamizar-isis-omp-00.txt>, Oct. 1998
63. J. Moy, *OSPF Version 2,* RFC 2178, Apr. 1998
64. E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms,* Computer Science Press, Pitman Inc., 1987
65. S. Baase, Computer Algorithms: Introduction to Design and Analysis, Addison-Wesley Publishing Company, 1987
66. Z. Wang, *Routing and Congestion Control in Datagram Networks,* Ph.D. thesis,

Dept. of CS., University College London, Jan. 1992

67. Z. Wang, *User-share differentiation (USD) scalable bandwidth allocation for differentiated services*, May 1998, Internet Draft.
68. Trillium Digital Systems, Inc. *Comparison of IP-over-SONET and IP-over-ATM Technologies*, November, 1997
69. *Controlling TCP/IP Bandwidth*, The Packeter Technical Forum, November 1998
70. *The New Paradigm for Differentiated Service Levels in Internet*, http://www.supranets.com/
71. *Cisco IOS Software Quality of Service Solutions, White Paper*, Cisco Systems, Inc. 1998
72. E. Rosen, A. Viswanathan and R.Callon, *Multi-Protocol Label Switching Architecture*, Internet draft<draft-ietf-mpls-arch-01.txt>, Mar. 1998
73. Wu-chang Feng, *Improving Internet Congestion Control and Queue Management Algorithms*, University of Michigan, 1999
74. S. Shenker, C. Partridge and R. Guerin, *Specification of Guaranteed Quality of Service*, RFC 2212, Sept. 1997
75. Shivkumar Kalyanaramen, *Re: Edge-to-edge Flow control draft: draft-shivkuma-ecn-diffserv-01.txt*, ECN archive message 00013
76. Ivy Hsu and Jean Walrand, *Dynamic Bandwidth Allocation for ATM Switches*, Feb.5, 1995
77. Edward W. Knightly, *Enforceable Quality of Service Guarantees for Busrsty Traffic Streams*, ECE Department, Rice University.
78. Raffaele Bolla, Franco Davoli and Mario Marchese, *Bandwidth Allocation and Admission in ATM Networks with Service Separaion, IEEE Communication Magazine*, May 1997
79. S. Keshav, *Congestion Control in Computer Networks*, Ph.D. thesis, U. of California, Berkeley, August 1991
80. S. Ohta and K. Sato, *Dynamic bandwidth control of the virtual path in an asynchronous transfer mode network*, IEEE Trans. Commun. 40, 7, 1239-1247, 1992

APPENDIX

A. Source Code for Service Level Negotiation

There are three programs in this package: agent.java, myserver.java, and myclient.java. They work cooperatively to perform Service Level Negotiation.

Agent.java plays the role of an Agent. It is stored at the Host side initially. When request comes, myclient.java will send this program as an Agent to the server. At the server side, agent.java compares user requests (maximum price and maximum queue length) with information in the server database and decides which service level to choose. There is one class in this program -- class "agent", whose algorithm is given in Chapter 5, page 83.

Myclient.java runs on the Host side. It collects information from the user, sets up FTP connection with the server, sends Agent to the server, and then reads responses from the server. There are three classes in this program: class "myclient", performing most of the tasks as a client; class "readLineThread", reading responses from the server; class "DataConn", transferring file using FTP.

Myserver.java resides at the server side as a service provider. It invokes the Agent and sends the result back to the Host. Class "myserver" is the only class in this program.

To be simplified, we use file queue.txt instead of a real database to store price and queue length information. Service provider should modify this file on a regular basis so that the up-to-date network states are reflected in this file. The numbers in the file represent, respectively, the price for A1 service, the queue length of A1, the price for A2 service, the queue length of A2, the price for A3 service, the queue length of A3, the price for A4 service and the queue length of A4.

A.1 agent.java

```
/*
 * @agent.java
 * @author: Suqiao Li
 * @date:  June, 1999
 * function: this program acts as an agent which negotiates with the
broker
 *          about service level
 *
 */
import java.io.*;
import java.lang.*;
import java.util.*;


public class agent {

public static DataInputStream input=null;

public static void main( String[] args) throws IOException
{ int i, j;


 DataInputStream input=null;
  String price="";
  String length="";
  int price_i=0; // price for service level
  int length_i=0;// occupied length of queue
  int arg_price=0;// maximum price the host want to pay
  int arg_length;// maximum queue length the host can accept

 StringTokenizer token=new StringTokenizer(args[0]);
        arg_price = Integer.parseInt(token.nextToken());
        token=new StringTokenizer(args[1]);
        arg_length = Integer.parseInt(token.nextToken());
        input = new DataInputStream(
                 new FileInputStream(new File("queue.txt")));

   price = input.readLine();//get A1 price
   while(price.compareTo("EOF") != 0)
    {   token=new StringTokenizer(price);
        price_i = Integer.parseInt(token.nextToken());
       if(price_i<(arg_price))// if A1 price is acceptable
       { length=input.readLine() ;
         token=new StringTokenizer(length);
         length_i = Integer.parseInt(token.nextToken());
         if(length_i<(arg_length))// if A1 length is acceptable
         {  System.out.println(" A1 Queue ");
       // return result: A1 service level
            break;
          }
          else
          { price=input.readLine();//get A2 price
            token=new StringTokenizer(price);
            price_i = Integer.parseInt(token.nextToken());
            if(price_i<(arg_price))//if A2 price is acceptable
            { length=input.readLine() ;
              token=new StringTokenizer(length);
              length_i = Integer.parseInt(token.nextToken());
              if(length_i<(arg_length))// if A2 length is acceptable
              {  System.out.println(" A2 Queue ");
             // retuen result: A2 service level
                 break;
               }
               else
               { price=input.readLine();//get A3 price
                 token=new StringTokenizer(price);
                 price_i = Integer.parseInt(token.nextToken());
                 if(price_i<(arg_price))// if A3 price is acceptable
                 { length=input.readLine() ;
                   token=new StringTokenizer(length);
                   length_i = Integer.parseInt(token.nextToken());
                   if(length_i<(arg_length))if A3 length is acceptable
                   {  System.out.println(" A3 Queue ");
                // return result: A3 service level
                      break;
                   }
                   else
                   { price=input.readLine();// get A4 price
                     token=new StringTokenizer(price);
                     price_i = Integer.parseInt(token.nextToken());
                     if(price_i<(arg_price)) // if A4 price is
acceptable
                      { System.out.println(" A4 Queue ");
                   // return result: A4 service level
                        break;
                       }
                      else
                      { System.out.println(" A3 Queue ");
                    // return result: A3 service level
                        break;
                       }
                    }
                 }
                 else
                 { System.out.println(" A2 Queue ");
                 // return result: A2 service level
                   break;
                 }
               }
            }
            else
            { System.out.println(" A1 Queue ");
           // return result: A1 service level
              break;
            }
       }
      }
      else
      { System.out.println("Best-effort Service");
     // return result: Best-effort service level
        break;
```

```
            }

        }//while
        input.close();

    } //main

    }//class agent
```

A.2 myserver.java

```
/*
 * @myserver.java
 * @author: Suqiao Li
 * @date:  June, 1999
 * function: this program is a server which acts as broker to negotiate
with
 *          agent about which service level to allocate.
 *
 *
 */

import java.io.*;
import java.lang.*;
import java.net.*;
import java.util.*;

public class myserver
{ private static Socket incoming;

  public static void main(String[] args) throws IOException
  { String response="";// response from the agent
    String str="e";
    String[] argument=new String[4]; // arguments sent to the agent

    try
    { ServerSocket s=new ServerSocket(8001);
      incoming=s.accept();
     }
     catch(Exception e){};
  // set up server socket, listen to request from host

    try
     { DataInputStream sin= new
DataInputStream(incoming.getInputStream());
       PrintStream sout= new PrintStream(incoming.getOutputStream());
       str=sin.readLine();
      }
    catch (Exception e)
    {System.out.println(e);}
    System.out.println("Read information from client...");
    argument[0]="java";
    argument[1]="agent";
    argument[2]=str.substring(0,(str.indexOf(";"))); //maximum price
    argument[3]=str.substring(str.lastIndexOf(";")+1); // maximum
length


    BufferedReader in= new BufferedReader(new InputStreamReader(
            Runtime.getRuntime().exec(argument).getInputStream()));
    System.out.println("Invoke agent.class...");
   // invoke agent

    try
    { DataInputStream sin= new
DataInputStream(incoming.getInputStream());
       PrintStream sout= new PrintStream(incoming.getOutputStream());
       while ((response= in.readLine()) != null)
     // get response from agent

       sout.println("Result of negotiation: "+response);
     // send back results to host

       System.out.println("Send back result...");
       incoming.close();
     // close the negotiation with agent
     }
     catch(Exception e)
     { System.out.println(e);}

    in.close();
  }//main
}//class myserver
```

A.3 myclient.java

```
/*
 * @myclient.java
 * @author: Suqiao Li
 * @date:  June, 1999
 *  function: this program runs on the host side. Read in the user's
 requests,
 *            and send agent to negotiate with the broker about the
 service
 *            level.
 *
 */

import java.io.*;
import java.net.*;
import java.lang.*;
import java.util.*;

public class myclient extends Thread
{ public static void main(String[] args) throws IOException
  { Socket socket = null;
    PrintWriter out = null;
    BufferedReader in = null;
    BufferedReader In = new BufferedReader(new
InputStreamReader(System.in));
    String str,str1,str2,str3;
    String fromServer= "";

    DataInputStream din= null;
    DataOutputStream dout= null;
    BufferedInputStream fin= null;
    BufferedOutputStream fout= null;

    //
    // collect requests from user
    //

    System.out.println("What is the maximun price you want to pay?:");
    str1 = In.readLine();
    System.out.println("What is the maximun queue length can you
accept?:");
    str2 = In.readLine();

   String response;
    Socket controlsocket = null;
    ServerSocket datasocket= null;
    BufferedReader controlin= null;
    PrintWriter controlout= null;


    //
    // connect to server
    //

    try {
      controlsocket= new Socket("SUSU", 21);
      controlin= new BufferedReader(new
InputStreamReader(controlsocket.getInputStream()));
      controlout= new PrintWriter(controlsocket.getOutputStream());
    // build up control connections for FTP transmition
    }
    catch (UnknownHostException ue) { System.err.println(ue);
System.exit(1); }


 new readLineThread(controlin).start();
    // retrieve greeting message from FTP server

   controlout.println("USER sqli");
   controlout.flush();
   new readLineThread(controlin).start();
   // login process

   controlout.println("PASS ******");
   controlout.flush();
   new readLineThread(controlin).start();
   // login process

   controlout.println("PORT 132,206,51,48,10,53");
   controlout.flush();
   new readLineThread(controlin).start();
   // send port number to server for active mode connection

   controlout.println("cwd li/server");
   controlout.flush();
   new readLineThread(controlin).start();
   // change directory to where server resides

   try {

      String filename="agent.class";

   //
   // Initiate Data Connection
   // active mode, use ServerSocket class
   //

    datasocket= new ServerSocket(2613);
    new DataConn(datasocket,  filename).start();

    controlout.println("TYPE I");
    controlout.flush();
    new readLineThread(controlin).start();
    // change to Binary mode

    //
    // Send FTP commands through Control Connection
    //

     try { sleep(5);} catch(InterruptedException ie) {;}
     controlout.println("STOR agent.class");
     controlout.flush();
```

```
      new readLineThread(controlin).start();
      //transmit the agent(agent.class) to the server



     try { sleep(5000);} catch(InterruptedException ie) {;}
     // wait for the transmission to complete
     }
     catch(IOException ie) {System.err.println(ie); }

     try
     { socket = new Socket("SUSU", 8001);
   //Create a socket which can communicate with port 8001(the server
port number)

       out = new PrintWriter(socket.getOutputStream(), true);
       in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
     }
     catch (IOException e)
     { System.out.println(e);
       System.exit(1);
     }


    out.println(str1+";"+str2);
// Send to the server
    System.out.println(" Waiting for result...");
    while ((str=in.readLine())!=null)
    {
      System.out.println(str);
   // Print out the response from the server

    }

     out.close();
     in.close();
     In.close();
     socket.close();
     controlout.close();
     controlin.close();
     controlsocket.close();
// Close the socket
 }
}

//
// class readLineThread is the thread which reads responses from server
//
class readLineThread extends Thread
{ BufferedReader in= null;

  public readLineThread(BufferedReader inn)
 {  super("Read Line Thread");
    this.in= inn;
  }
 // constructor
```

```
  public void run()
  { String fromServer= "";
    try
    {   while ((fromServer= in.readLine()) != null)
            System.out.println(fromServer);
    }
    catch (IOException ie) {System.exit(1);}
  }
}//class readLineTread


//
// class DataConn is the thread which sends files to server via FTP
//
class DataConn extends Thread
{ ServerSocket socket;
  Socket datasocket;
  int type;
  String file, target;
  boolean client;

 public DataConn(ServerSocket s, String f)
 {
    super("Data Connection");
    this.socket= s;
    this.file= f;
    this.client= false;
  }
  // constructor

  public void run()
  {
    DataInputStream din= null;
    DataOutputStream dout= null;
    BufferedInputStream fin= null;
    BufferedOutputStream fout= null;
    int data= -1;

    try
     {
       datasocket= socket.accept();

       dout= new DataOutputStream(datasocket.getOutputStream());
       fin= new BufferedInputStream(new FileInputStream(file));

       while ((data= fin.read()) >= 0)
         dout.write(data);

       fin.close();
       dout.close();

       datasocket.close();
       socket.close();
    }

    catch (FileNotFoundException fnfe)
   {
          System.out.println("**** File not found. Try correct file name.");
          return;
        }
        catch (IOException ie) {System.err.println(ie); }
    }
  }
```

A.4 queue.txt (simplified database of prices and occupation percents for different levels)

```
100
50
200
30
300
20
400
10
EOF
```

B. Source Code for Bandwidth Negotiation

There are seven programs in this package: agent1.java, agent2.java, agent3.java, myserver.java, myclient1.java, myclient2.java, and myclient3.java. They work cooperatively to perform Bandwidth Negotiation. The last three of these programs represent three different hosts, while the first three are the three Agents sent by these Hosts. Since there is no essential difference between these Hosts and Agents, we list only one pair of them here.

Agent1.java plays the role of an Agent. It is stored at the Host side initially. When a request comes, myclient1.java will send this program as an Agent to the server. At the server side, agent1.java gets the token price from the Broker and modifies its request according to that new price. The negotiation between Agent and Broker may last several rounds before a final consensus is reached. There is one class in this program -- class "agent", whose algorithm is described in Chapter 5, page 86.

Myclient1.java runs on the host side. It collects information from the user, sets up the FTP connection with the server, sends the Agent to the server, and then reads responses from the server. There are three classes in this program: class "myclient1", performing most of the tasks as a client; class "readLineThread", reading responses from the server; class "DataConn", transferring file using FTP.

Myserver.java resides at the server side as a service provider. It collects requests from all the Agents and modifies token price according to the ratio of total requests to available token number. The negotiation process continues until that ratio converges to 1. Class "myserver" is the only class in this program. The algorithm for this class is given in Chapter 5, page 88.

B.1 agent1.java

```
/*
 * @agent1.java
 * @author: Suqiao Li
 * @date:  June, 1999
 * function: this program acts as an agent which negotiates with the
broker
 *           about how many tokens to buy.
 *
 */



import java.io.*;
import java.lang.*;
import java.util.*;


public class agent1 {

public static DataInputStream input=null;

public static void main( String[] args) throws IOException
{ int i, j;
  int org_price;
  // Original token price from the broker
  int price;
  // New token price form the broker
  int max_price;
  // Maximum price which the host want to pay
  int pre_bw;
  // Preferred bandwidth of the host
  int min_bw;
  // Minimum bandwidth of the host
  int quit_bd=0;
  // The request reduced to 0 when token price exceeds the maximum
price
  int cache;


  //
  // collect information from the broker
  //
  StringTokenizer token=new StringTokenizer(args[0]);
  max_price = Integer.parseInt(token.nextToken());
  token=new StringTokenizer(args[1]);
  pre_bw = Integer.parseInt(token.nextToken());
  token=new StringTokenizer(args[2]);
  min_bw = Integer.parseInt(token.nextToken());
  token=new StringTokenizer(args[3]);
  org_price = Integer.parseInt(token.nextToken());
  token=new StringTokenizer(args[4]);
  price = Integer.parseInt(token.nextToken());
  if(price>max_price)
 // if token price greater than the maximum acceptable price
  {  System.out.println(Integer.toString(quit_bd));
 // ask for 0 token ( quit this round negotiation)
  }
  else
  {  cache=pre_bw*org_price/price;
    // change request according to the change of the price
     if(cache>min_bw)
    // if the new request is greater than the minimum
     {  pre_bw=cache;
        System.out.println(Integer.toString(pre_bw));
    // send the new request to the broker
     }
     else System.out.println(Integer.toString(pre_bw));
    // maintain the original request
   }

} //main

}//class agent
```

B.2 myserver.java

```
/*
 * @myserver.java
 * @author: Suqiao Li
 * @date:  June, 1999
 * function: this program is a server which acts as broker to negotiate
with
 *           three different agents about how many tokens to buy.
 *
 */

import java.io.*;
import java.lang.*;
import java.net.*;
import java.util.*;
import java.lang.Integer.*;

public class myserver
{ private static Socket[] incoming=new Socket[4];

  public static void main(String[] args) throws IOException
  { String response=""; // the response from the agents
    String clienti="";
    String str="", cache="";
    String[][] argument=new String[4][7];
    // arguments sent from the broker to the agents
    int tr=0;
    //Total Request
    int atn=100;
    //Available Token Number
    int price=10;
    // Price for one token ( initially $10)
    int[] request={0,0,0,0};
    // Requests from each client
    int[] port={0,8001,8002,8003};
    //Port number for agents to access server
    StringTokenizer token;
    ServerSocket[] s=new ServerSocket[4];
    // serversocket for different ports

   //
   // get initial information: preferred bandwidth, minimum bandwidth
and
   // maximum price from three agents. Put these information into
different
   // arrays.
   //
    for(int i=1; i<4; i++)
    {  try
       { s[i]=new ServerSocket(port[i]);
         incoming[i]=s[i].accept();
       }
       catch(Exception e){};
       try
```

```
        { DataInputStream sin= new
DataInputStream(incoming[i].getInputStream());
          PrintStream sout= new
PrintStream(incoming[i].getOutputStream());
          str=sin.readLine();
         }
        catch (Exception e)
        {System.out.println(e);}
        System.out.println("Read information from client "+i);
        cache=str.substring(0,(str.indexOf(";")));

        if(cache.endsWith("Client1"))
      // if the information is sent by host1
        {  argument[1][0]="java";
           argument[1][1]="agent1";
           argument[1][2]=str.substring(0,(str.indexOf("C")));

argument[1][3]=str.substring((str.indexOf(";")+1),str.lastIndexOf(";"))
;
           argument[1][4]=str.substring(str.lastIndexOf(";")+1);
           argument[1][5]=Integer.toString(price);
         }
         else if(cache.endsWith("Client2"))
      // if the information is sent by host2

               {  argument[2][0]="java";
                  argument[2][1]="agent2";
                  argument[2][2]=str.substring(0,(str.indexOf("C")));

argument[2][3]=str.substring((str.indexOf(";")+1),str.lastIndexOf(";"))
;
                  argument[2][4]=str.substring(str.lastIndexOf(";")+1);
                  argument[2][5]=Integer.toString(price);
               }
               else
       // if the information is sent by host3
                    { argument[3][0]="java";
                      argument[3][1]="agent3";

argument[3][2]=str.substring(0,(str.indexOf("C")));

argument[3][3]=str.substring((str.indexOf(";")+1),str.lastIndexOf(";"))
;

argument[3][4]=str.substring(str.lastIndexOf(";")+1);
                      argument[3][5]=Integer.toString(price);
                    }
     }//for


   //
   // sum up total initial requests
   //
   for(int i=1;i<4;i++)
   {  token=new StringTokenizer(argument[i][3]);
      request[i] = Integer.parseInt(token.nextToken());
      tr=tr+request[i];
     }

   //
   // Negotiation process going on until tr/atn converges to 1
   //

     while(((float)(tr/atn)<0.9)||((float)(tr/atn)>1))
     { argument[1][5]=Integer.toString(price);
       argument[2][5]=Integer.toString(price);
       argument[3][5]=Integer.toString(price);
    // argument[][5] is the price of last round

       price=price*tr/atn;
     // change price according to the ratio of tr/atn

       argument[1][6]=Integer.toString(price);
       argument[2][6]=Integer.toString(price);
       argument[3][6]=Integer.toString(price);
     // argument[][6] is the price of this round

       tr=0;

     //
     // invoke all the agents again, and collect new requests from them
     //
       for(int i=1; i<4;i++)
       { BufferedReader in= new BufferedReader(new InputStreamReader(
             Runtime.getRuntime().exec(argument[i]).getInputStream()));
         //System.out.println("Invoke agent.class...");
         try
         { DataInputStream sin= new
DataInputStream(incoming[i].getInputStream());
           PrintStream sout= new
PrintStream(incoming[i].getOutputStream());

           response=in.readLine();
         }
         catch(Exception e)
           { System.out.println(e);}

          token=new StringTokenizer(response);
          request[i] = Integer.parseInt(token.nextToken());
          tr=tr+request[i];
          in.close();
        }

     }// while

     //
     // return the results to hosts.
     //
     for(int i=1;i<4;i++)
     { try
       { DataInputStream sin= new
DataInputStream(incoming[i].getInputStream());
         PrintStream sout= new
PrintStream(incoming[i].getOutputStream());
```

```
        //  while ((response= in.readLine()) != null)

          sout.println("Result of negotiation: ");
          sout.println("    Token number: "+ request[i]);
          sout.println("    Price: "+price);
          System.out.println("Send back result...");
          incoming[i].close();
        }
      catch(Exception e)
      { System.out.println(e);}

     }

  }//main
}//class myserver
```

B.3 myclient.java

```
/*
 * @myclient1.java
 * @author: Suqiao Li
 * @date:  June, 1999
 * function: this program runs at the host side. It reads information
 from user * and send out agent to negotiate with broker.
 *
 */

import java.io.*;
import java.net.*;
import java.lang.*;
import java.util.*;

public class myclient1 extends Thread
{ public static void main(String[] args) throws IOException
  { Socket socket = null;
    PrintWriter out = null;
    BufferedReader in = null;
    BufferedReader In = new BufferedReader(new
InputStreamReader(System.in));
    String str,str1,str2,str3;
    String fromServer= "";

    DataInputStream din= null;
    DataOutputStream dout= null;
    BufferedInputStream fin= null;
    BufferedOutputStream fout= null;


    //
    // Collect information from hosts
    //
    System.out.println("What is the maximun price you want to pay?:");
    str1 = In.readLine();
    System.out.println("What is the preferred bandwidth do you
want?:");
    str2 = In.readLine();
    System.out.println("What is the minimun bandwidth can you
accept?:");
    str3=In.readLine();


  String response;
    Socket controlsocket = null;
    ServerSocket datasocket= null;
    BufferedReader controlin= null;
    PrintWriter controlout= null;


  //
  // connect to server
  //
```

```
   try {
       controlsocket= new Socket("PEACH", 21);
       controlin= new BufferedReader(new
InputStreamReader(controlsocket.getInputStream()));
       controlout= new PrintWriter(controlsocket.getOutputStream());
     // build up control connections for FTP transmition
     }
     catch (UnknownHostException ue) { System.err.println(ue);
System.exit(1); }


    new readLineThread(controlin).start();
   // retrieve greeting messages from FTP server

    controlout.println("USER sqli");
    controlout.flush();
    new readLineThread(controlin).start();
  // login process

    controlout.println("PASS ******");
    controlout.flush();
    new readLineThread(controlin).start();
  // login process

    controlout.println("PORT 132,206,51,47,10,53");
    controlout.flush();
    new readLineThread(controlin).start();
  // send port number to server for active mode connection

    controlout.println("cwd li/bw/server");
    controlout.flush();
    new readLineThread(controlin).start();
  // change directory to where server resides

      try {

        String filename="agent1.class";

    //
    // Initiate Data Connection
    // active mode, use ServerSocket class
    //

      datasocket= new ServerSocket(2613);
      new DataConn(datasocket,  filename).start();

      controlout.println("TYPE I");
      controlout.flush();
      new readLineThread(controlin).start();
    // change to Binary mode


   //
   // Send FTP commands through Control Connection
   //
```

```
     try { sleep(5);}
     catch(InterruptedException ie) {;}
     controlout.println("STOR agent1.class");
     controlout.flush();
     new readLineThread(controlin).start();
   // transmit the agent(agent1.class) to the server

     try { sleep(5000);}
     catch(InterruptedException ie) {;}
    }//try
     catch(IOException ie)
         {System.err.println(ie); }

     try
     { socket = new Socket("PEACH", 8001);
    // Create a socket which can communicate with port 8001( server port
number)

       out = new PrintWriter(socket.getOutputStream(), true);
       in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
     }
     catch (IOException e)
     { System.out.println(e);
       System.exit(1);
     }

    str1=str1.concat("Client1");
   // mark the string for identification

    System.out.println("str1= "+str1);
    System.out.println(" send out: "+str1+";"+str2+";"+str3);
    out.println(str1+";"+str2+";"+str3);
// Send to the server
     System.out.println(" Waiting for result...");
     while ((str=in.readLine())!=null)
// Print out the response from the server
    {
      System.out.println(str);
    }

     out.close();
     in.close();
     In.close();
     socket.close();
     controlout.close();
     controlin.close();
     controlsocket.close();
// Close the socket
 }
}


//
// class readLineThread is the thread which reads responses from server
//
```

```
class readLineThread extends Thread
{
  BufferedReader in= null;

  public readLineThread(BufferedReader inn)
 {
    super("Read Line Thread");
    this.in= inn;
  }
  // constructor

  public void run()
  {
    String fromServer= "";
    try
    {
       while ((fromServer= in.readLine()) != null) {
        System.out.println(fromServer);
      }
    }
    catch (IOException ie) {System.exit(1);}
  }
}//class readLineTread


//
// class DataConn is the thread which transfers file using FTP
//

class DataConn extends Thread
{
  ServerSocket socket;
  Socket datasocket;
  int type;
  String file, target;
  boolean client;

  public DataConn(ServerSocket s, String f)
 {
    super("Data Connection");
    this.socket= s;
    this.file= f;
    this.client= false;
  }
  // constructor

  public void run()
  {
    DataInputStream din= null;
    DataOutputStream dout= null;
    BufferedInputStream fin= null;
    BufferedOutputStream fout= null;
    int data= -1;

    try
    {
       datasocket= socket.accept();
```

```
          dout= new DataOutputStream(datasocket.getOutputStream());
          fin= new BufferedInputStream(new FileInputStream(file));

          while ((data= fin.read()) >= 0)
          dout.write(data);

          fin.close();
          dout.close();




      datasocket.close();
      socket.close();
    }
    catch (FileNotFoundException fnfe)
    {
      System.out.println("**** File not found. Try correct file name
again.");

      return;
    }
    catch (IOException ie)
      {System.err.println(ie); }
 }
}
```