차별화 서비스를 제공하는 IP 네트워크의 Bandwidth 사용량 관리를 위한 분산 Edge-to-Edge Throughput 모니터링 기법

2002

김 재 영

박 사 학 위 논 문

차별화 서비스를 제공하는 IP 네트워크의 Bandwidth 사용량 관리를 위한 분산 Edge-to-Edge Throughput 모니터링 기법

김 재 영 (金 宰 瑩)

전자 컴퓨터 공학과 (네트워크 전공)

포항공과대학교 대학원

2002

차별화 서비스를 제공하는 IP 네트워크의 Bandwidth 사용량 관리를 위한 분산 Edge-to-Edge Throughput 모니터링 기법

Distributed Edge-to-Edge Throughput Monitoring Methods to Manage Bandwidth Usage in IP Networks Supporting Differentiated Services

Distributed Edge-to-Edge Throughput Monitoring Methods to Manage Bandwidth Usage in IP Networks Supporting Differentiated Services

by

Jae-Young Kim
Department of Electrical and Computer Engineering
Pohang University of Science and Technology

A thesis submitted to the faculty of Pohang University of Science and Technology in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Electrical and Computer Engineering.

Pohang, Korea
October 22, 2001

Approved by

_____

Major Advisor: James Won-Ki Hong

차별화 서비스를 제공하는 IP 네트워크의 Bandwidth 사용량 관리를 위한 분산 Edge-to-Edge Throughput 모니터링 기법

김 재 영

위 논문은 포항공과대학교 대학원 박사 학위논문으로 학위논문 심사위원회를 통과하였음을 인정합니다.

2001년 10월 22일

학위논문심사위원회 위원장 홍 원 기 (인)

위원 강 교 철 (인)

위원 이 재 용 (인)

위원 김 종 (인)

위원 서 영 주 (인)

DECE 19983119 김재영, Jae-Young Kim, Distributed Edge-to-Edge Throughput Monitoring Methods to Manage Bandwidth Usage in IP Networks Supporting Differentiated Services, 차별화 서비스를 제공하는 IP 네트워크의 Bandwidth 사용량 관리를 위한 분산 Edge-to-Edge Throughput 모니터링 기법, Department of Electrical and Computer Engineering, 2002, 114P, Advisor: James Won-Ki Hong, Text in English.

ABSTRACT

The Differentiated Services (DiffServ) framework has been proposed by the IETF as a simple service structure that can provide different Quality of Service (QoS) to different classes of users in IP networks. IP packets are classified into one of a limited number of service classes, and are marked in the packet header for easy classification and differentiated treatments when transferred within a DiffServ network domain. The DiffServ framework defines simple and efficient QoS differentiation mechanisms for the Internet. However, the current DiffServ concepts do not provide a complete QoS management framework. Since traffic flows in IP networks are unidirectional from one network point to another, and routing path and traffic demand of the traffic flows are changing constantly, it is important to monitor end-to-end traffic status, as well as traffic status in a single node. This thesis proposes two distributed throughput monitoring methods that collect the statistical data of each service class in every DiffServ router and estimate edge-to-edge throughput and drops of the aggregated IP flows by combining routing topology and traffic status. A formal definition and modeling of edge-to-edge DiffServ flows and methods for concatenating edge-to-edge throughput and drops are presented. The first method calculates the minimum and maximum amount of edge-to-edge flows from local statistics. The second method calculates the exact amount of edge-to-edge flows by marking an edge ID on packet headers at ingress edge routers and counting the marked packets at egress

edge routers. When the amount of edge-to-edge flows is calculated, the QoS status of the edge-to-edge flows at each transit router on the routing path is also derived with simple throughput concatenation rules. Discrete simulation of sample DiffServ networks is performed to validate the monitoring methods. It is demonstrated that the Random Early Detection (RED) queue used in DiffServ networks is appropriate for the proposed monitoring methods. These methods are applied to several practical DiffServ management functions to show the applicability and effectiveness of the methods. Next, an SNMP-based edge-to-edge DiffServ flow management system is designed and implemented to realize useful service management functionality using the proposed monitoring methods. Monitoring edge-to-edge DiffServ flows enables network administrators to manage the bandwidth usage of the aggregated DiffServ flows more effectively and can be a useful building block for sophisticated DiffServ management functionalities.

Contents

List of Figures

List of Tables

1. Introduction

The explosive growth of the Internet has resulted in an exponential increase in the number of users and the amount of network traffic. However, network bandwidth is frequently insufficient to satisfy the exponential increase in bandwidth requirements from various network applications. The quality of network service has been degraded at points where network bandwidth becomes scarce. Unexpected packet loss, delay, and jitter occur when many network packets compete for insufficient network bandwidth at bottleneck points. Since both people and applications are becoming increasingly dependent on network services, to guarantee a sufficient amount of service quality has become a natural requirement for every user. Limitations on providing service differentiation in IP networks have been addressed in many ways [20, 21, 22, 23].

In order to provide service differentiation on the Internet, the Internet Engineering Task Force (IETF) has recently introduced Differentiated Services (DiffServ) framework [1]. DiffServ is an alternative approach to Integrated Services (IntServ) [2] because IntServ relies on per-flow states and per-flow processing in every network node, which is difficult to deploy in large backbone networks. Instead, DiffServ controls the aggregation of traffic flows; that is, DiffServ flows, at each routing decision point without using a signaling protocol. IPv4 Type-of-Service (ToS) octet or IPv6 traffic class octet is used for distinguishing the DiffServ flows [3]. Since DiffServ is a simpler and more scalable solution for Internet backbone networks, DiffServ is widely accepted as a feasible solution for providing Internet QoS [24].

DiffServ applies administrative domain concepts. At the boundary of the DiffServ domain, edge routers perform classification of traffic flows based on 5-tuple information, composed of source and destination IP addresses, a type of transport protocol, and source and destination application port numbers. The edge routers perform marking on the most-significant six bits of ToS fields of incoming

packets. This 6-bit mark is called Differentiated Services Code Point (DSCP). Within one domain, core routers forward traffic according to the DSCP value of DiffServ flows. Since the edge routers have already marked the DSCP value of the incoming traffic, core routers need not handle complex information in such traffic. Core routers perform various differentiating actions, such as dropping, metering, shaping, and remarking, according to different DSCP values. This different packet treatment performed at each router is called Per-Hop Behavior (PHB). Best-Effort (BE), Expedited Forwarding (EF) [4], and Assured Forwarding (AF) [5] are examples of proposed PHBs from the IETF.

However, current DiffServ specifications have limitations in providing complete QoS management framework [25]. As described in Figure 1, QoS management tasks are composed of repeated cycles of configuration, monitoring, and analysis. Without any one component of the cycle, QoS management cannot be complete. From the point of view of QoS management, current DiffServ RFCs and drafts are mainly for QoS provisioning and configuration only. QoS monitoring and analysis based on the measurement results are not yet addressed in detail.

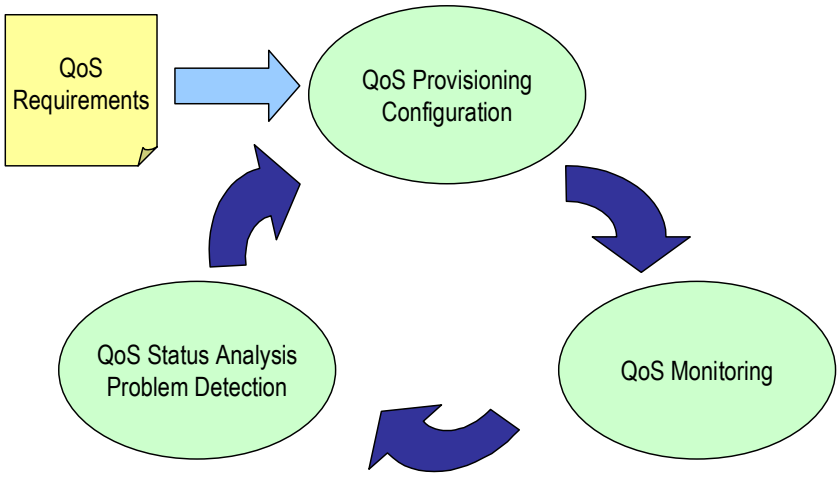


Figure 1. QoS Management Cycle

Moreover, DiffServ does not provide mechanisms to control end-to-end services though real network QoS is meaningful when the QoS at each routing path is controlled in detail. Within a DiffServ domain, network packets in a certain service class are treated equally no matter what their source and destination. In reality, packets with a different source and destination use a different routing path within a DiffServ domain and they consume different portion of reserved bandwidth. To provide finer-grained network service and real end-to-end network QoS, DiffServ needs additional mechanisms.

Managing DiffServ networks requires numbers of complicated functions interconnected and cooperated with each other. In this chapter, several related work has been reviewed and explained. Since the DiffServ framework needs efficient provisioning mechanisms with correct monitoring on network status and lacks complete management architecture composed of configuration, monitoring, and analysis, it is noted that efficient and scalable monitoring on DiffServ network status is important when developing DiffServ management systems.

Monitoring real-time network traffic provides a set of basic and primitive operations to extract correct status information on monitoring points. In order to manage the monitoring information more efficiently, the SNMP framework has been introduced to build manger-agent architecture in monitoring systems. SNMP-based monitoring systems enable manager systems to collect and analyze large statistical data from numbers of network monitoring points and thus to report more advanced network status to human administrators. Since most network services are organized on network paths encompassing multiple numbers of network nodes, it becomes important to know end-to-end network status between two specific network points. Two end-to-end monitoring methods (active and passive) have pros and cons to each other and include different measurement and analysis mechanisms.

Current management efforts on DiffServ networks can be divided into three categories. The first category is configuration management using SNMP-based DiffServ MIB and COPS-based DiffServ PIB. Two different configuration

management approaches are being standardized in IETF independently. The second category is monitoring using DSMON MIB. Each DiffServ router with DSMON MIB installed is able to collect statistical information of network status for each DiffServ QoS class. The third category is about management architecture. High-level management architectures containing all aspects of management functionality are being studied and suggested. However, the current management efforts still need more research work for providing efficient and scalable monitoring methods in DiffServ networks.

This thesis tries to suggest useful monitoring methods that help DiffServ management system to understand current DiffServ network status and to make appropriate management decisions on DiffServ components. Current DiffServ framework separates routing information from QoS control information. It is noted that combining two sets of information will provide valuable data for understanding DiffServ network status. Concatenating local QoS monitoring following the edge-to-edge routing path can extract QoS status of edge-to-edge DiffServ flows. The extracted edge-to-edge QoS information of each DiffServ class can be used for efficient and scalable management functions. Edge-to-edge QoS information is essential when providing sophisticated QoS management operations. To build edge-to-edge QoS information, locally-observed QoS information is aggregated by following routing paths. Aggregated QoS information of DiffServ flows can provide detailed network-wide traffic status. Locating overloaded links and suggesting possible solutions can be drawn from the QoS status of DiffServ flows.

In these methods, each DiffServ router in a DiffServ domain monitors QoS information of multiple QoS classes for every network interface. Distributed monitoring has benefits in using less network resources and providing more efficient solutions. When monitoring network QoS information, we can have two types of measurement methods: active and passive measurements [27]. Our measurement method is passive, which does not alter or affect network traffic.

The research objectives of following chapters are listed below.

- To provide a concrete definition of edge-to-edge DiffServ flows, and end-to-end behavior of the flows in DiffServ networks.
- To suggest a set of QoS monitoring and aggregation methods for constructing edge-to-edge QoS of DiffServ flows by combining a routing topology and traffic status measured at each DiffServ routers.
- To validate the suggested methods in a simulated environment with various combinations of routing topology and traffic demand.
- To show how to build a working system for managing QoS of DiffServ flows in a real DiffServ testbed composed of a set of Linux DiffServ routers and Ethernet connections.

The remainder of this thesis is organized as follows. Chapter 2 summarizes current Internet QoS frameworks suggested in detail and discusses integration and management approaches of the Internet QoS frameworks. Chapter 3 explains in more detail about the DiffServ framework and then reveals limitations in providing end-to-end QoS control mechanisms. From Chapter 4, the proposed mechanisms for monitoring edge-to-edge DiffServ flows are introduced and explained. Simple modeling of the edge-to-edge DiffServ flow is suggested and edge-to-edge QoS aggregation rules by marking identifier of each source edge router in network packets. In addition, various examples of using edge-to-edge DiffServ flow information are described and verified in Chapter 5. In Chapter 6, a prototype implementation of an edge-to-edge DiffServ flow management system working on a DiffServ testbed is configured with a set of DiffServ Linux routers using the proposed QoS monitoring methods and SNMP framework. In Chapter 7, various related work is surveyed and compared. Finally, Chapter 8 summarizes the pros and cons of the proposed methods and their contributions. Possible future research directions for further extension of this research are also noted.

2. Internet QoS Frameworks

Quality of Service (QoS) is an well-established term in computer literature. Many research areas, such as real-time systems, distributed multimedia systems, communication networks, etc., define and utilize different meanings of QoS from different viewpoints. Supporting appropriate QoS in the research areas has been challenging work, with many problems to solve. There have been various QoS frameworks to solve the QoS problems [28] and also approaches to define formal meanings of QoS support [29].

Recently, QoS support in IP networks has become important for two reasons. One reason is that the Internet, which is based-on TCP/IP technology, is hugely popular today and traffic volume is too huge to meet user requirements in every network link. The other reason is that the current best-effort IP network does not yet provide any service differentiation mechanism while the network bottleneck links degrade the end-to-end service quality due to unexpected packet loss, jitter, and delay. QoS in IP networks can be defined as the concept that transmission rates, error rates, and other characteristics of data communication can be measured, improved, and, to some extent, guaranteed in advance [30]. Difficulties and problems in supporting QoS in IP networks have been addressed in many research papers [20, 21, 22, 23] and extensively explained in various technical books [30, 31, 32, 33, 34].

Specification of IP protocol initially includes a QoS provisioning mechanism by defining a Type of Service (ToS) byte in the IP header. The ToS provides an indication of the abstract parameters of the quality of service desired. These parameters are to be used to guide the selection of the actual service parameters when transmitting a datagram through the particular network [37]. However, until recently, the ToS byte was not necessarily considered important because controlling the ToS byte was an overhead to network nodes.

In order to overcome current problems in provisioning IP QoS, the IETF first

standardized the Integrated Services (IETF) framework [2] and then proposed the Differentiated Service (DiffServ) framework [1] after noticing the IntServ has a scalability problem as described in Chapter 1. The DiffServ finally uses the ToS byte for distinguishing different classes of packets meeting the initial IP philosophy.

Multiprotocol Label Switching (MPLS) is recently proposed as an enabling technology to support QoS between layer 2 and layer 3 protocols [38, 39, 40, 41]. MPLS can establish and maintain different routing paths for different priority classes of packets so that efficient traffic engineering and QoS provisioning is possible in MPLS domains.

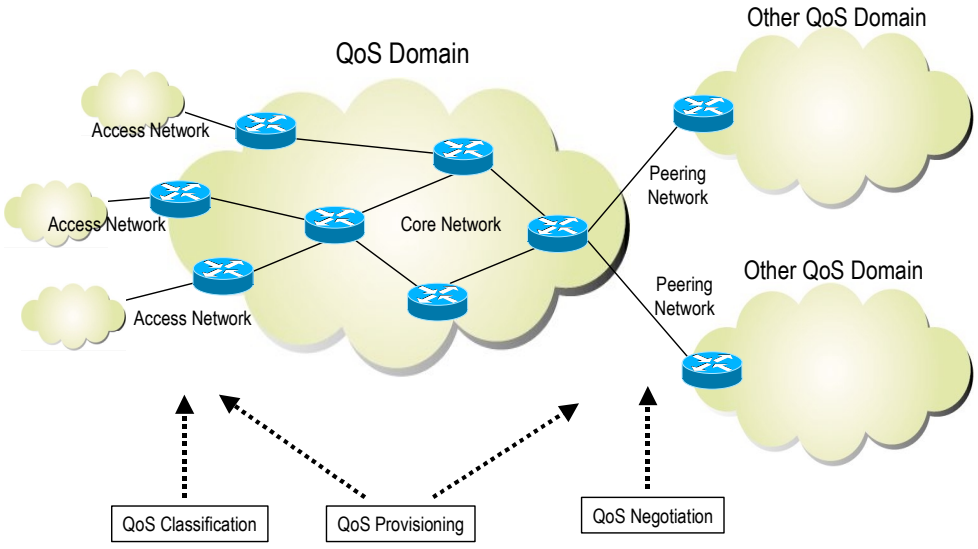


Figure 2. General IP QoS Network Architecture

General IP QoS network framework common to three different IP QoS frameworks is depicted in Figure 2. A QoS domain consists of a set of routers connected each other with a consistent QoS provisioning policies. Users of IP QoS services reside in access networks belong to the QoS domain. For each access network the QoS domain distinguishes incoming traffic according to

predefined classification rules. The classified IP packets are forwarded differently within the QoS domain and thus service differentiation is achieved. There might be more than one QoS domain existing. Between two different QoS domains resides peering network. Within this network domain, different QoS parameters are negotiated and determined to adjust service qualities provided from each QoS domain.

Different IP QoS frameworks (IntServ, DiffServ, and MPLS) complement each other. They can exist together and play different roles in integrated environment. Methods for combining IntServ and DiffServ [42, 43], DiffServ and MPLS [44], and all of them [45] have been studied. The future Internet QoS framework will integrate various features from the three frameworks.

In this chapter, brief discussions about features of three different Internet QoS frameworks currently being suggested from the IETF are explained and compared. In addition, integration approaches for different QoS frameworks are described.

2.1 Integrated Services (IntServ)

The IETF set up the IntServ working group in 1994 to expand the Internet's service model to better meet the needs of emerging, diverse voice / video applications. It aims to clearly define the new enhanced Internet service model as well as to provide the means for applications to express end-to-end resource requirements with support mechanisms in routers and subnet technologies.

In the IntServ framework [2], three classes of service are proposed based on applications delay requirements. The first is guaranteed service [50], with bandwidth, bounded delay, and no-loss guarantees. The second is controlled load service [51], which approximates best-effort services in a lightly loaded network. The third is best-effort service, similar to that which the Internet currently provides under a variety of load condition.

The main point is that the guaranteed service and controlled load classes are

based on quantitative service requirements, and both require signaling and admission control in network nodes. These services can be provided either per-flow or per-flow-aggregate, depending on flow concentration at different points in the network.

The service guarantee needs bandwidth reservations. IntServ uses Resource ReserVation Protocol (RSVP) for its internal reservation processes [52, 53, 54]. RSVP assumes that resources are reserved for every router hop in the path between receiver and transmitter using end-to-end signaling. This, in turn, requires flow-specific state in the routers, which represents an important and fundamental change to the Internet model. It also provides a way to communicate the application's requirements to network elements along the path, and to convey QoS management information between network elements and the applications.

The major advantage of IntServ is that it provides service classes which closely match the different application types described earlier and their requirements. For example, the guaranteed service class is particularly well suited to the support of critical, intolerant applications. On the other hand, critical, tolerant applications and some adaptive applications can generally be efficiently supported by controlled load services. Other adaptive and elastic applications are accommodated in the best-effort service class.

Another strength is guaranteed delay bounds. In [55], it is showed that if the router implements a Weighted Fair Queuing (WFQ) scheduling discipline, and if the nature of the traffic source can be characterized, then there will be an absolute upper bound on the network delay of the traffic in question. The simple and very powerful result applies not just to one switch, but to general networks of routers.

A major characteristic of IntServ is that it leaves the existing best-effort service class mostly unchanged (except for a further subdivision of the class), so it does not involve any change to existing applications. This is an important property since IntServ is then capable of providing this class of service as efficiently as the current Internet. IntServ also leaves the forwarding mechanism in the network unchanged. This allows for an incremental deployment of the framework, while

allowing end systems that have not been upgraded to support IntServ to be able to receive data from any IntServ class (with, of course, a possible loss of guarantee).

The most problematic issue for IntServ concerns the scalability of RSVP, especially in high-speed backbone networks. The amount of resources that a router needs for RSVP processing and storage increases proportionally with the number of QoS flows. Traffic measurements show that most end-to-end IP connections are very short-lived, and that there are several thousands active connections at any time in a backbone router. Consequently, numerous IntServ flows on a high-bandwidth link place an excessive burden on routers. Furthermore, if a topology change occurs, the reservations must be renegotiated simultaneously.

Another problem is that the Internet architecture was founded on the concept that all flow-related state should be in the end systems, although designing the TCP/IP protocol suite on this concept led to a robustness which is central to its success. The flow state added to the routers for resource reservation can be softened to preserve the robustness of the Internet protocol suite. RSVP sends periodic refresh messages to maintain the state along the reserved paths. In the absence of a timely refresh message the state automatically times out and is deleted.

2.2 Differentiated Services (DiffServ)

In 1998, the DiffServ working group was formed by the IETF. DiffServ is a bridge between IntServ's guaranteed QoS requirements and the best-effort service offered by the Internet today. The DiffServ framework [1] aims at providing simple and scalable service differentiation. It does this by discriminating and treating the data flows according to their traffic class, thus providing a logical separation of the traffic in the different classes.

To overcome the limitation of IntServ with RSVP, DiffServ enhancements to IP enable scalable service discrimination in the Internet without the need for per-

flow state and signaling at every hop. A variety of services may be built from a small well-defined set of building blocks that are deployed in network nodes. The services may be either end-to-end or intradomain.

At their boundaries, service providers build their offered services with a combination of traffic classes (to provide controlled unfairness), traffic conditioning (a function that modifies traffic characteristics to make it conform to a traffic profile and thus ensure that traffic contracts are respected), and billing (to control and balance service demand). Provisioning and partitioning of both boundary and interior resources are the responsibility of the service provider and, as such, outside the scope of DiffServ. For example, DiffServ does not impose either the number of traffic classes or their characteristics on a service provider.

Unlike RSVP, no QoS requirements are exchanged between the source and the destination, eliminating the inherent setup costs associated with RSVP. Short-lived flows benefit from DiffServ because the absence of QoS setup costs improves responsiveness and reduces the overhead required for a quick discussion with another host.

If each packet conveyed across a service provider's network simply carries in its header an identification of the traffic class (called a DiffServ Code Point, DSCP) to which it belongs, the network can easily provide a different level of service to each class. It does this by appropriately treating the corresponding packets, say, by selecting the appropriate per-hop behavior (PHB) for each packet. In both IPv4 and IPv6, the traffic class is denoted by the use of the DS header field.

It must be noted that DiffServ is based on local service agreements at customer/provider boundaries. Therefore, end-to-end services will be built by concatenating such local agreements at each domain boundary along the route to the final destination. The concatenation of local services to provide meaningful end-to-end services is still an open research issue.

The only functionality actually imposed by DiffServ in interior routers is packet classification. This classification is simplified from that in RSVP because it

is based on a single IP header field containing the DSCP, rather than multiple fields from different headers. This has the potential of allowing functions performed on every packet, such as traffic policing or shaping, to be done at the boundaries of domains, so forwarding is the main operation performed within the provider network.

Another advantage of DiffServ is that the classification of the traffic, and the subsequent selection of a DSCP for the packets, need not be performed in the end systems. Indeed, any router in the stub network where the host resides, or the ingress router at the boundary between the stub and provider networks, can be configured to classify (on a per-flow basis), mark, and shape the traffic from the hosts. Such routers are the only points where per-flow classification may occur, which does not pose any problem because they are at the edge of the Internet, where flow concentration is low. The potential noninvolvement of end systems, and the use of existing and widespread management tools and protocols allows swift and incremental deployment of the DiffServ framework.

To provide several services with differing qualities within the same network simultaneously is a very difficult task. Despite its apparent simplicity, DiffServ does not make this task any simpler. Instead, in DiffServ it was decided to keep the operating mode of the network simple by pushing as much complexity as possible onto network provisioning and configuration. Of course, network provisioning and configuration have been performed since the creation of the very first communication networks, and thus they benefit from long experience, and available tools and traffic models. However, so far, large networks have mainly offered a single type of service (best-effort service in the Internet, interactive voice in telephone networks, etc.). The provisioning of networks providing multiple classes of service at the same time is therefore a new area which requires much research to consider the added complexity due to possible adverse interactions between different classes of service. The construction of end-to-end services by concatenating local service agreements is also a nontrivial research issue.

The key to provisioning is the knowledge of traffic patterns and volumes traversing each node of the network. This also requires a good knowledge of network topology and routing. The problem with the Internet is that provisioning will be performed on a much slower timescale than the timescales at which traffic dynamics and network dynamics (e.g., route changes) occur. This problem can be illustrated with the simplest case of a single service provider network whose service agreements with customers are static. Although the amount of traffic entering the domain is known and policed, it is impossible to guarantee that overloading of resources will be avoided. This is caused by two factors:

- The entering packets can be bound to any destination in the Internet, and may thus be routed towards any border router of the domain (except the one where it entered). In the worst case, a *substantial proportion* of the entering packets might all exit the domain through the same border router
- Route changes can suddenly shift vast amounts of traffic from one router to another.

Therefore, unless resources are massively overprovisioned in both interior and border routers, traffic and network dynamics can cause momentary violation of service agreements, especially those relating to quantitative services. On the other hand, massive overprovisioning results in a very poor statistical multiplexing gain, and is therefore inefficient and expensive.

To increase resource usage in their network, service providers can trade generality and robustness for efficiency. For example, to limit the amount of expensive resources dedicated to the support of quantitative services, service providers can limit quantitative service contracts to apply between any pair of border routers in the domain. In such a case, the service would apply only to packets entering the domain at a designated ingress router and leaving the domain at a designated egress router. This helps solve the first problem described above at the cost of generality, since only packets bound for destinations “served” through the egress router can benefit from the service. Of course, to ensure that the egress router is in the route to any given destination, the interdomain routing entry for

that destination must be statically fixed in the ingress router. Even for a fixed ingress-egress pair, intradomain routing dynamics can still occur. This means that the set of internal routers visited by the packets traveling between the ingress and egress routers can still suddenly change. However, the "directionality" of the traffic considered here is such that the number of possible routes is considerably reduced compared with the general case, and so is the resulting and necessary overprovisioning. A service provider could, however, reduce to a minimum the overprovisioning of quantitative services offered between pairs of border routers by "pinning" the intradomain route between those routers. Fixing the egress router for a given destination and/or pinning internal routes between border routers nevertheless incurs a loss of robustness. In multicast, where receivers can join and leave the communication at any time, the problem of efficient provisioning will be more severe.

Alternatively, a service provider may wish to use dynamic logical provisioning and configuration (i.e., sharing of resources between classes) as an answer to the problems of network and traffic dynamics. However, depending on the type of service agreement (qualitative, relative, or quantitative) and the QoS parameters involved in the agreement, dynamic logical provisioning might require signaling and admission control.

From the point of view of a flow, the class bandwidth is not a meaningful parameter. Indeed, bandwidth is a class property *shared* by all the flows in the class, and the bandwidth received by an individual flow depends on the number of competing flows in the class as well as the fairness of their respective responses to traffic conditions in the class. Therefore, to receive some quantitative bandwidth guarantees, a flow must "reserve" its share of bandwidth along the data path, which involves some form of end-to-end signaling and admission control (at least among logical entities called *bandwidth brokers*). This end-to-end signaling should also track network dynamics (i.e., route changes) to enforce the guarantees, which can prove to be very complex. Furthermore, even qualitative bandwidth agreements require end-to-end signaling and admission control. This is because

even if one class is guaranteed to have more bandwidth than another, the number and behavior of flows in the latter class may result in smaller shares of bandwidth for these flows than for flows in the other class. Hence, in this case, end-to-end signaling would also be required to ensure that, in every node along the path, the bandwidth received by a flow in a high bandwidth class is greater than the bandwidth received by a flow in a smaller bandwidth class.

On the other hand, delay and error rates are class properties that *apply* to every flow of a class. This is because in every router visited, all the packets sent in a given class share the queue devoted to that class. Consequently, as long as each router manages its queues to maintain a relative relationship between the delay and/or error rate of different classes, relative service agreements can be guaranteed without any signaling. However, if quantitative delay or error rate bounds are required, end-to-end signaling and admission control are also required.

End-to-end signaling and admission control would increase the complexity of the DiffServ framework. The idea of dynamically negotiable service agreements has also been suggested as a way to improve resource usage in the network [1]. Such dynamic service-level agreements would require complex signaling, since the changes might affect the agreements a provider has with several neighboring networks. The timescale on which such dynamic provisioning could occur would be limited by scalability considerations, which in turn could impede its usefulness.

It should be noted that from the point of view of complexity, a DiffServ scenario with dynamic provisioning and admission control is very close to an IntServ scenario with flow aggregation. The difference is that precise delay and error rate bounds might not be computed with DiffServ, since the delays and error rates introduced by each router in the domain may not be available to the bandwidth broker.

Although DiffServ will undoubtedly improve support for a number of applications and is urgently needed in the Internet, it does not represent the ultimate solution for QoS support for all types of applications. The suitability of DiffServ for a given application could also depend on the context in which that

application is used. As an example, with Internet telephony, we see that DiffServ is suitable for providing a cheap solution for internal calls between remote sites of a company by emulating leased lines between these sites. However, DiffServ may prove unsuitable for the support of telephony over the Internet for the general public, because people do not usually restrict their calls to only a few destinations.

More detailed DiffServ mechanisms and limitations are explained in Chapter 4.

2.3 Multiprotocol Label Switching (MPLS)

Multiprotocol Label Switching (MPLS) is an IETF standard for a new switching paradigm that enables packet switching at layer 2 while using layer 3 forwarding information. Thus, MPLS combines the high-performance capabilities of layer 2 switching and the scalability of layer 3-based forwarding. The term multilayer routing covers approaches to the integration of layer 3 datagram forwarding and layer 2 switching that extend beyond the use of the techniques found within gigabit routing/switching. The approach uses label lookups to allow more efficient packet classification, and the potential to engineer the network and manage the impact of data flows. A flurry of vendor-specific approaches to multilayer routing appeared between 1994 and 1997, including Ipsilon's IP Switching [56], Toshiba's Cell Switch Router (CSR) [57], IBM's ARIS [58], Cisco's Tag Switching [59], and IPSOFACTO [60]. The fact that these approaches were proprietary, and in the main produced incompatible solutions, led to the formation of the Internet Engineering Task Force (IETF) Multi Protocol Label Switching working group in 1997.

The MPLS working group is addressing the issues of the scalability of routing, the provision of more flexible routing services, increased performance, and more simplified integration of layer 3 routing and circuit-switching technologies, with the overall goal of providing a standard label-swapping architecture [61].

Each MPLS packet has a header that is either encapsulated between the link

layer and the network layer, or resides within an existing header, such as the virtual path/channel identifier (VPI/VCI) pair within Asynchronous Transfer Mode (ATM). At most, the MPLS header contains a 20-bit label, TTL field, Exp (experimental) field, stack indicator, next header type indicator, and checksum as in Figure 3 [63].

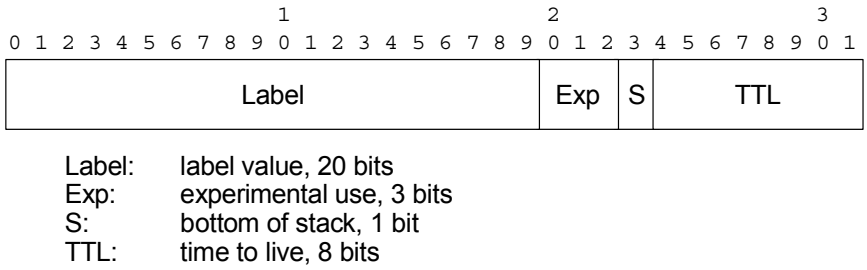


Figure 3. MPLS Label Stack Entry Format

MPLS defines a fundamental separation between the grouping of packets that are to be forwarded in the same manner (the forwarding equivalence classes, or FECs), and the labels used to mark the packets. This is purely to enhance the flexibility of the approach. At any one node, all packets within the same FEC could be mapped onto the same locally significant label (given that they have the same requirements). However, there are instances where one may wish to engineer the network in such a way that several different labels are used (e.g., when wishing to explicitly differentiate between streams). The assignment of a particular packet to an FEC is completed once, at the entry point to the network. MPLS-capable routers (Label-Switched Routers, LSRs) then use only the label and CoS field in order to make packet forwarding and classification decisions. Label merging is possible where multiple incoming labels are to receive the same FEC.

MPLS packets are able to carry a number of labels, organized in a last-in first-out stack as in Figure 4. This can be useful in a number of instances, such as where two levels of routing take place across transit routing domains. Regardless

of the existence of the hierarchy, in all instances the forwarding of a packet is based on the label at the top of the stack. In order for a packet to travel through a tunnel, the node at the transmitting side of the tunnel pushes a label relating to the tunnel onto the stack, and sends the packet to the next hop in the tunnel.

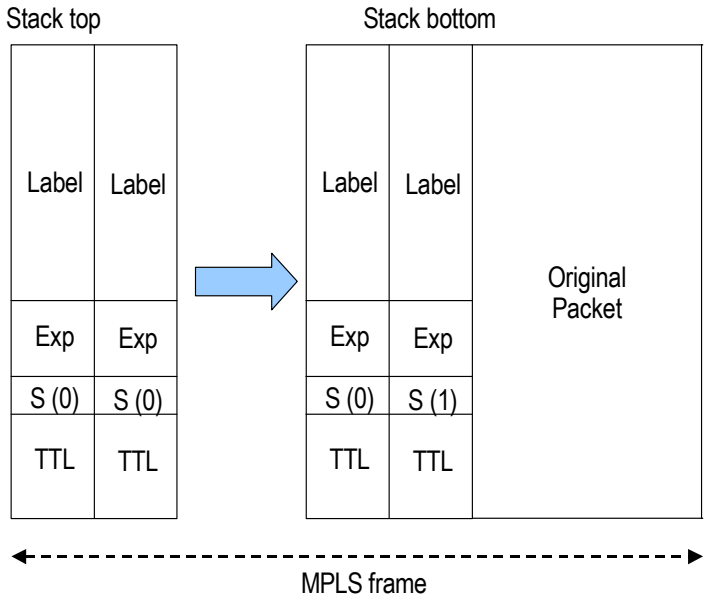


Figure 4. MPLS Label Stacking

A collection of LSRs are combined to make a Label-Switched Path (LSP). Two options are defined for the selection of a route for a particular forwarding class. Hop-by-hop routing defines a process where each node independently decides the next hop of the route. Explicit routing is where a single node (often the ingress node of a path) specifies the route to be taken (in terms of several or all of the LSRs in the path). Explicit routing may be used to implement network policies, or allow traffic engineering in order to balance the traffic load.

MPLS is able to work in an environment that uses any data link technology, connection-oriented and connectionless. MPLS also provides the potential for all traffic to be switched, but this depends on the granularity of the label assignment, which again is flexible and depends on the approach used to identify traffic.

Labels may be assigned per address prefix (e.g., a destination network address prefix) or set of prefixes, and can also represent explicit routes. On a finer-grained level, labels can be defined per host route and also per user. At the lowest level, a label can represent a combined source and destination pair, and in the context of RSVP can also represent packets matching a particular filter specification.

MPLS needs a mechanism for distributing labels in order to set up paths. The framework does not assume a single protocol (known as a Label Distribution Protocol, LDP [62]) to complete this task. MPLS label distribution requires reliability and the sequencing of messages that relate to a single FEC. While some approaches use protocols that sit directly over IP (thus implying they are unlikely to be able to meet these reliability requirements), a number of the defined LDPs solve this issue by operating over TCP.

MPLS-capable devices are able to provide additional functionality beyond the best-effort packet forwarding found within a gigabit router. This flexibility means that in principle it is possible to support ideas such as QoS differentiation. The fundamental separation between forwarding class and label assignment provides a great deal of flexibility. While packets within a class are to be processed in the same way, this approach means that traffic can be engineered to varying extents.

Explicit routing (a subset of constraint-based routing) allows the specification of the route to be taken across the network. This is enabled within MPLS by allowing a label to represent a route, without the overhead of source routing found within normal IP forwarding (making it too resource-intensive for use in most circumstances). Different paths can be selected in order to allow traffic engineering to be carried out effectively, allowing network loads to be balanced in a far more flexible manner than manually configuring virtual circuits (as with other primitive approaches to engineering IP traffic). The engineering of paths in such a way implies a simple mechanism for measuring traffic between edge network devices making use of an LSP.

In Internet service provider (ISP) environments, where service differentiation is likely to mean users will be charged in terms of the network QoS exploited, the

ability within the MPLS framework to specify per-host and per-user label assignment is likely to be very useful for billing purposes.

While the label stack concept provides benefits, the idea of having packets carry a number of labels is likely to increase overheads, in terms of making the MPLS header larger. With topology-driven label assignment (where labels are allocated and distributed without reference to the traffic), a full mesh of labels will be established. The overhead of this approach is essentially relative to the size of the network, and has the potential to use a vast number of labels. This can be a large overhead in instances when labels are allocated to routes where very little traffic is flowing.

In terms of the provision of varying levels of QoS, MPLS poses a number of issues. Label assignment based on support for traffic flows will require a path to be created when the flow is detected, therefore implying that some latency exists before a full path is created. In this instance, the overhead will increase in relation to the number of flows being supported and the duration of the flows. Label assignment in order to support short flows implies a large overhead. When label distribution is included as part of a reservation protocol (e.g., RSVP), the overheads and scalability of such a protocol must also be considered.

The ability of MPLS to support a number of link-layer technologies provides a high degree of flexibility. However, in terms of the provision of connections with a level of associated QoS, mechanisms are required to ensure that the QoS specified for an LSP is maintained by the underlying link layer. This may not be possible in some instances (e.g., with an Ethernet) where firm guarantees cannot be made (because of the inherent nature of the technology). Where ATM technology is used with MPLS, in most instances the LDP acts as the ATM signaling protocol. This implies that a low-level control protocol is required which is able to configure connections with defined levels of QoS. While work is progressing in this area within the IETF General Switch Management Protocol (GSMP) Working Group [85, 86], wide-scale support for this type of protocol by major switch vendors is not yet evident.

2.4 Integration of Different Internet QoS Frameworks

Recent research work on providing IP QoS focuses on integration of existing IP QoS frameworks because those IP QoS frameworks complement each other. Each QoS framework has different roles and appropriate fields of its own. To provide real end-to-end QoS guarantee connecting application endpoints, those QoS frameworks need to be combined and integrated.

Before explaining details of integration approaches, Table 1 summarizes and compares the characteristics of IP QoS frameworks previously explained.

Table 1. Comparison of Different IP QoS Frameworks

Framework	IntServ	DiffServ	MPLS
Network Model	edge / core	edge / core	edge / core
Granularity	fine	coarse	flexible
Resource Reservation	RSVP	open issue	open issue
Routing Path	fixed	not fixed	multiple
Classification	per microflow	per DSCP	per LSP
Services	Guaranteed Controlled-Load	Expedited Forwarding Assured Forwarding	Olympic Services (Gold/Silver/Bronze)
Roles	access network	backbone network	traffic engineering

Integration of IntServ and DiffServ has been proposed in [63] and discussed in various research work [42, 43, 64, 65]. The key idea is a model in which peripheral subnetworks are RSVP- and IntServ-aware and these subnetworks are interconnected by DiffServ networks. In this model, the scalability of DiffServ networks extends the reach of IntServ/RSVP networks. Intervening DiffServ networks appear as a single RSVP hop to the IntServ/RSVP networks. RSVP

signaling messages are carried transparently through the DiffServ networks. Devices at the boundaries between the IntServ/RSVP networks and the DiffServ networks process the RSVP messages and provide admission control based on the resource availability within the DiffServ network.

Integrating DiffServ over MPLS network has been studied in [44]. This research work modifies LDP so that the MPLS network understands the DSCP of packet headers and provides DiffServ-related QoS differentiation over ATM MPLS networks. This method forms a simple and efficient Internet model capable of providing applications with differentiated QoS. The need for complex IP and ATM signaling protocols like RSVP and P-NNI can be eliminated. No per-flow state information is required leading to increased scalability. A lightweight signaling protocol like LDP with the appropriate extensions along with the ATM traffic management mechanisms provide all the necessary functionality and flexibility required by large networks in a simple manner and without sacrificing precious resources.

Finally, an approach to integration of all three IP QoS framework has been proposed [45]. An integration model called DRUM (DiffServ and RSVP/IntServ Use of MPLS) is suggested to deliver end-to-end service guarantees for both DiffServ and IntServ networks, where part of the underlying technology used for IP transport is MPLS, using DiffServ-like mechanisms for QoS provision. Mechanisms for mapping appropriate DiffServ and IntServ service classes into the set of MPLS service classes have been proposed. Since per-flow state information and complex scheduling is not required in MPLS domain, the scalability gain increased by integrating different QoS frameworks.

In summary, different IP QoS frameworks have pros and cons when deployed. Thus in order to provide end-to-end QoS guarantees and flexible controls in heterogeneous Internet environment, different IP QoS frameworks should be integrated and coordinated in various ways. One QoS technology cannot replace another QoS technology totally. Each IP QoS framework has different roles and benefits to be harmonized with other IP QoS frameworks.

3. DiffServ Framework and Its Limitations

In the previous chapter, three different IP QoS frameworks are explained with different features and characteristics. Among them, DiffServ is especially important to Internet service providers. Since the DiffServ enables Internet service providers to construct scalable QoS solutions in comparatively large Internet backbone networks, it is essential to deploy DiffServ networks within the domain of those Internet service providers. In this chapter, DiffServ mechanisms are reviewed in more detail to explain how the framework operates in detail. However, in spite of its simplicity and scalability, real deployment of DiffServ networks cannot be easily found out yet. There are a few limitations of the DiffServ framework from the management point of view. This chapter reveals how those limitations are raised. This thesis tries to solve these limitations for managing DiffServ network domains.

3.1 DSCP, PHB, and Traffic Aggregation

DiffServ proposes a simple and scalable method to differentiate a set of traffic among network nodes. The method is based on a simple model where traffic entering a network is classified and possibly conditioned at the boundaries of the network, and assigned to different behavior aggregates. Each behavior aggregate is identified by a single Differentiated Services Code Point (DSCP).

DSCP is the most-significant 6 bits from the IPv4 Type-of-Service (ToS) octet or IPv6 traffic class octet. This 6-bit field indicates how each router should treat the packet. This treatment is called Per-Hop Behavior (PHB). PHB defines how an individual router will treat an individual packet when sending it over the next hop through the network. Being 6 bits long, the DSCP can have one of 64 different binary values. Four types of PHBs have been defined as proposed standards thus

far [6, 7, 8, 9]. They are default, class-selector, Assured Forwarding (AF), and Expedited Forwarding (EF) PHBs. Table 2 summarizes the standard PHBs and DSCP values. Among 64 available DSCP values 21 values are reserved. The other DSCP values can be used for provisioning locally configurable mappings.

Table 2. Standard PHBs and Reserved DiffServ Code Points

PHB name	DSCP	description
default	000000	best-effort (RFC 1821)
class-selector	xxx000	7 classes (RFC 2474)
AFxy	xxxyy0	4 classes with 3 drop probabilities (RFC 2597)
EF	101110	no drop (RFC 2598)

	000	001	010	011	100	101	110	111
000	default							
001	class-selector		AF11		AF12		AF13	
010			AF21		AF22		AF23	
011			AF31		AF32		AF33	
100			AF41		AF42		AF43	
101							EF	
110								
111								

- bits in the first column: first three bits in DSCP
- bits in the first row: last three bits in DSCP

DiffServ-enabled network nodes handle classes of network packets differently by using the DSCP value in the packet header within an administrative domain. The DiffServ domain is a contiguous set of DiffServ-enabled nodes, which operates with a common service provisioning policy and a set of PHB groups implemented within each node. To select a PHB from one of the PHB groups

supported within the domain, edge routers classify (and possibly condition) ingress traffic to ensure that packets crossing the domain are appropriately marked. Core routers check only the DSCP value of forwarded packets and perform predefined PHB actions on the packets. It requires no per-flow state in backbone and trunk routers. This internal behavior saves time and resources for providing different service quality to different classes of users.

3.2 DiffServ Router Architecture

A DiffServ domain consists of a set of DiffServ routers under a set of consistent provisioning rules. There are two types of DiffServ routers, one is edge router located at the boundary of the DiffServ domain and the other is core router interconnecting edge routers. Each DiffServ router is configured to handle DSCP-marked packets with pre-defined Per-Hop Behaviors (PHBs).

A DiffServ router is a fundamental DiffServ-enabled network node. The conceptual model and requirements of the DiffServ routers are discussed in [13, 14]. A DiffServ router is considered to have a routing module, a set of Traffic Control Blocks (TCBs), a queuing module, and a configuration and monitoring module as shown in Figure 5.

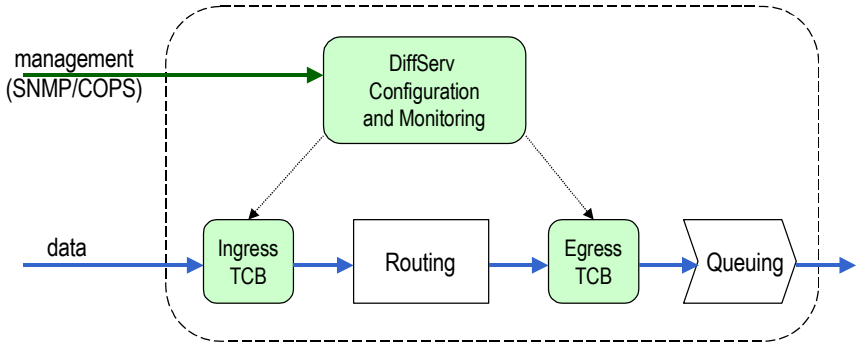


Figure 5. Conceptual Model of a DiffServ Router

DiffServ-related modules are separated from the routing module to simplify the addition of the DiffServ capability to the existing router. Traffic conditioning can be performed either at the ingress point or at the egress point, or both. At each ingress or egress point, a set of TCBs is cascaded to form complicated traffic shaping. The queuing module is a set of underlying packet queues that store packets before a router sends them out. The management module for the DiffServ router can be operated in several ways, such as SNMP or Common Open Policy Service (COPS) protocol [15, 16]. The management module configures TCB parameters and monitors the performance of each TCB.

A DiffServ-enabled network node has a cascaded set of traffic conditioning blocks (TCB) for handling DSCP-marked network packets. A traffic conditioning block is a minimum logical element that controls DiffServ packets passing through the DiffServ network node. It receives packets from the network and classifies them into a predefined set of traffic aggregates by looking up the DSCP value in packet headers. Each traffic aggregate is metered, marked, shaped, or dropped separately. Figure 6 illustrates four components of a traffic conditioning block [3].

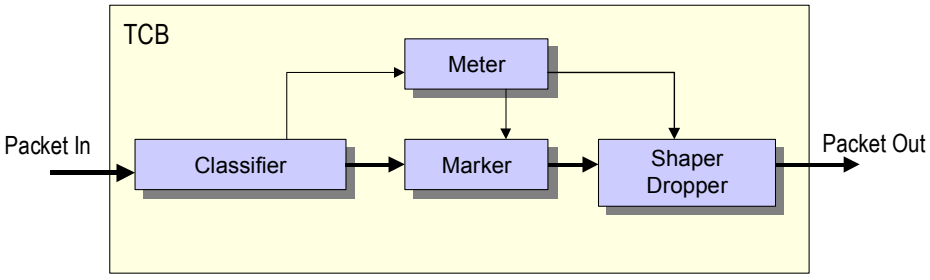


Figure 6. Basic Traffic Conditioning Block of DiffServ

A classifier selects network packets in a traffic stream based on the content of some portion of the packet header. Five-tuple information and DSCP value in packet headers are used for the classification process. A meter measures the temporal properties of the stream of packets selected by a classifier. It passes state information to other conditioning actions to trigger a particular action for each

packet. A marker sets the DSCP of a packet and a shaper delays some or all of the packets in a traffic stream in order to bring the stream into compliance with a traffic profile. A dropper may discard some or all of the packets in a traffic stream for the same reason.

More than one traffic conditioning block may exist in a DiffServ network node. In this case, each TCB is connected to each other; that is, they are cascaded so that a packet is conditioned by one TCB and then sent to another TCB to take the next conditioning operation. This mechanism enables a DiffServ network node to build much more flexible controls.

A conceptual TCB can be modeled as illustrated in Figure 7. When a packet comes in, the packet is classified by the classifier with the predefined classification rules. Classified packets are sent to one of multiple priority queues and affected by the packet drop algorithms within each queue when congestion occurs. The scheduler picks a packet from one of the priority queues by following scheduling algorithms. Traffic flows following the procedures are shaped and tailored to meet the requirements of the given traffic class.

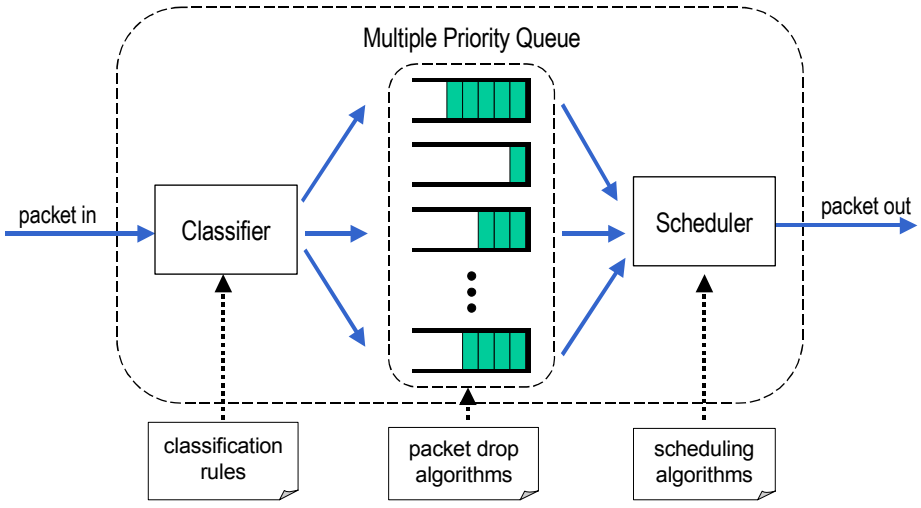


Figure 7. Conceptual Model of a TCB

3.3 DiffServ Service Models

The IETF DiffServ working group defines a few DiffServ service models and associated PHBs for the service models. Each DiffServ service is constructed by configuring each PHBs on DiffServ routers [30, 33].

3.3.1 Class Selector Service

Limited backward compatibility with the old Ipv4 ToS field's precedence classes [26] is achieved by reserving part of pool 1 for class selector PHBs. RFC 2474 defines the Class Selector PHBs as those that closely approximate the packet handling of various IPv4 precedence levels. The associated Class Selector codepoints are DSCP values in the range 000000 to 111000 (represented as xxx00). A router implementing Class Selector PHBs must implement at least two, and it may implement up to eight distinct PHBs. Multiple Class Seclector Codepoints may map to a single Class Selector PHBs.

RFC 2473 has the following to say about the handling rules for Class Selector PHBs.

- PHBs selected by a Class Selector codepoint should give packets a probability of timely forwarding that is not lower than that given to packets marked with a Class Selector codepoint of a lower relative order, under reasonable operating conditions and traffic loads.
- PHBs selected by codepoints 11x000 must give packets a referential forwarding treatment by comparison to the PHB selected by codepoint 000000 to preserve the common usage of IP precedence values 110 and 111 for routing traffic.
- PHBs selected by distinct Class Selector codepoints should be independently forwarded; that is, packets marked with different Class Selector codepoints may be re-ordered.

Various queuing and scheduling algorithms (WFQ, WRR, CBQ, Priority, and so on) meet these requirements. It is sufficient that a router is configured to properly map a packet arriving with any given Class Selector codepoint onto a PHB that meets the preceding requirements.

3.3.2 Expedited Forwarding (EF) Service

RFC 2598 defines the very simple EF PHB and its associated service model. An EF PHB requests every router along the path to always service EF packets at least as fast than the rate at which EF packets arrive. This situation leads to three subsequent requirements.

- Rate shape or police EF traffic on entry to the DS domain, to cap the rates at which EF traffic may enter the network core.
- Configure the EF packet-serving interval at every core router to exceed the expected aggregate arrival rate of EF traffic.
- EF packet-servicing intervals must be unaffected by the amount of non-EF traffic waiting to be scheduled at any given instant.

RFC 2598 notes that it is possible to meet the scheduling requirements with various scheduling algorithms, but does not mandate any particular approach. Jitter characteristics are the main sources of difference.

Although in practice EF packets will be sent to a specific queue for appropriate scheduling, the definition of EF service is such that on average this queue ought to be small or empty. As consequences, the EF PHB is a suitable building block for low-loss, low-latency, and low-jitter edge-to-edge services. RFC 2598 notes that EF can be used to build ‘virtual lease line’ services because it carves out a guaranteed slice of edge-to-edge bandwidth protected against all other users of the DiffServ domain.

By definition, packet drops within the network are meant to be rare for EF service. Individual traffic flows using EF service are rate-shaped or aggressively policed on entry to the DiffServ domain. Thus, correctly configured core routers should be free from EF traffic arriving with an aggregate rate exceeding their configured service intervals. As such, progressive or statistical drop mechanisms such as RED are not part of the EF PHB definition.

A single DSCP of 101110 is recommended to indicate EF PHB. It is worth noting that although the PHB is easy to define, an actual service based on EF requires careful coordination of policing, shaping, and scheduler service intervals along any path EF traffic is likely to take.

3.3.3 Assured Forwarding (AF) Service

Defined in RFC 2597, AF is actually a PHB group for edge-to-edge services specified in terms of relative bandwidth availability and multi-tiered packet drop characteristics. Whereas EF supports services with 'hard' bandwidth and jitter characteristics, the AF group allows more flexible and dynamic sharing of network resources, which supports the 'soft' bandwidth and loss guarantees appropriate for bursty traffic. RFC 2597 actually defines four PHB groups, each independently supporting AF behavior.

Two distinct classification contexts are encoded within the DSCP; a packet's service class and drop precedence. The service class provides the context with which to select an appropriate queue. The drop precedence provides context to weight RED-like behavior [66] expected from the queue manager, making it more or less aggressive, depending on whether a packet's drop precedence is high or low. AF requires active queue management with independent RED-like behavior on each queue to keep long-term congestion down while allowing short-term burstiness.

We can consider the DSCP to be laid out an *nnnmm0* with *nnn* encoding the

service class (queue selection) and *mm* encoding the drop precedence. RFC 2597 currently defines four service classes and three levels of drop precedence. A shorthand notion exists for referring to a particular AF PHB. AFxy refers to AF service class x with drop precedence y. Table 2 gives the DSCP values associated with each combination of service class and drop precedence.

Although an AF DSCP identifies one of four queues, it does not specify a queue's maximum size or the scheduler service interval associated with each queue. The network operator configures these parameters on a case-by-case basis. Each service class is distinguished by the level of forwarding resources it receives at each hop, independent of the other three classes. To prevent possible re-ordering of packets belonging to application flows within a service class, an AF-compliant router must not map different service classes into the same queue and is not allowed to distribute packets belonging to a single service class across multiple queues.

Specific drop probabilities for each precedence level are assigned by the network operator to meet the desired packet-loss characteristics for each class. The only requirements are that drop precedence 3 must have a more aggressive drop probability than precedence 2 has, and that precedence 2 must be more aggressive than precedence 1 has. In addition, the probability functions are per-class. Packets marked AF12 may be subject to an entirely different drop probability function than packets marked AF22. Although RFC 2597 defines three drop precedence levels, a minimal AF implementation may get by with only two distinct drop probability functions. In this case, both precedence 2 and 3 are mapped to the function returning the higher drop probability. As with EF, actual edge-to-edge service based on an AF PHB group requires coordination between edge routers (to limit the type of traffic mapped to each AF class and drop precedence) and the core routers (to ensure that appropriate resources and behaviors are provided to each class and drop precedence).

3.4 Limitations of DiffServ Framework

Thus far, the detailed functionality of DiffServ framework has been explained. DiffServ achieves scalability while providing coarse-grained differentiation. The IETF DiffServ working group defines a set of documents describing detailed operations of the DiffServ framework. However, DiffServ framework has several intrinsic limitations. In this section, limitations of DiffServ framework are revealed.

DiffServ's architectural simplicity is attractive, but actual deployment of edge-to-edge services requires filling in many unspecified parameter values, and few guidelines exist for network operators to follow. Mapping hundreds or thousands of mostly unrelated microflows to a limited set of PHBs requires a careful balancing act – an act that is as much dependent on the network's topology at any point in time as it is on the ingress traffic conditioning and interior queuing and scheduling.

Two approaches are available to ensure that individual edge-to-edge services survive being aggregated as they pass through the core:

- Aggressively rate shape or police at the edges
- Overprovisioning the core

In the first case, the network as a whole appears fairly intolerant from the perspective of the eternal traffic sources. In the second case, the network will, on average, be underutilized. For example, when provisioning EF service class in core routers, each core routers reserves bandwidths of the amount of one EF service class times number of edge routers at worst case. These conditions represent the two competing issues facing network designers – balancing service flexibility against network efficiency.

The provision of differentiated services requires careful network wide provisioning and configuration. Provisioning refers to the determination and allocation of the resources needed at various points in the network. Provisioning

may dictate the addition or removal of physical resources at various points (physical provisioning). Provisioning may also dictate the modification of operating parameters within existing physical network equipment to alter the relative share of the equipment's resources which are allotted to one or another class of traffic (logical provisioning). Configuration refers to the distribution of the appropriate operating parameters to network equipment to realize the provisioning objectives.

The second limitation is that DiffServ reserves resources per traffic aggregate, not per microflow. When provisioning DiffServ networks, each DiffServ router reserves network resources to each DiffServ class it can handle. This might cause some greedy microflows to consume most of network bandwidths and make some microflows experiences starvation for resources. Current DiffServ framework this unfair resource consumption cannot be identified and resolved.

The third limitation concerns providing end-to-end QoS. Since DiffServ network achieves QoS differentiation by configuring PHBs in each DiffServ router and each DiffServ router controls DiffServ traffic separately from routing control, it is said that the concatenation of PHB configuration following the routing path cannot give correct end-to-end QoS information because at each DiffServ router, DiffServ packets within the same DiffServ class traverse different routing path. Thus, the routing path of a DiffServ class can be merged and splitted while DiffServ packets come across DiffServ network domain. There is no way to provide status of correct end-to-end DiffServ QoS.

Common reasons for those limitations of DiffServ network come from lack of concrete management functionality in DiffServ network. Especially, current DiffServ network needs to have a well-defined set of QoS monitoring methods for extracting valuable status information for provisioning DiffServ network more effectively. Constant monitoring QoS of traffic flows from one edge router to another edge router is considered as important when controlling DiffServ QoS in careful ways.

4. Distributed Edge-to-Edge Throughput Monitoring

In this chapter, the main ideas for thesis research and research methodologies for supporting the ideas are explained. First, several assumptions for the throughput monitoring methods are explained to clarify conditions validating suggested methods. Next, concrete modeling of edge-to-edge DiffServ flows are presented and two different throughput monitoring methods for the edge-to-edge DiffServ flows are suggested. Discrete simulation with the ns-2 network simulator [19] verifies the suggested methods and several considerations when implementing suggested methods in IP networks are discussed.

The concept of de-aggregating DiffServ flows according to its source and destination edge enables network operators to analyze and understand the service status in more detail. Network operators can obtain finer-grained service control from coarse-grained DiffServ networks.

4.1 Assumptions for Monitoring Methods

There are several assumptions that should be explained before the presentation of detailed edge-to-edge throughput monitoring methods in DiffServ networks. With these assumptions, the proposed monitoring methods are valid and possible to be realized.

The first set of assumptions are related to IP routing. The primitive IP routing policy is destination-based single-path routing. Each DiffServ router is assumed to operate in accordance with this IP routing policy. At a certain point of time, the DiffServ router keeps a routing table that contains IP subsets of destination addresses and addresses of next hop routers for the destination subsets. When we have the routing tables from all the DiffServ routers in a DiffServ domain, we can easily construct a routing topology at that time. Routing path from one edge router

to the other edge router is assumed unique at each monitoring interval. Of course, the routing path can be changed dynamically. Various reasons such as link outages, node repairs, and load balancing can cause the entries of routing tables to be modified constantly. In order to prevent the routing updates from breaking consistency of edge-to-edge monitoring results, it is assumed that the routing updates should be reported to the edge-to-edge monitoring system so that the edge-to-edge throughput can be re-calculated.

The next assumption for monitoring edge-to-edge throughput monitoring methods is that each DiffServ router is assumed to keep track of statistics of transmitted packets belong to each DiffServ class. The number of transmitted packets and the number of dropped packets are examples of such statistic information. These per-class statistics should be stored in the DiffServ routers and reported to the edge-to-edge monitoring system periodically. As explained in Figure 7, the TCBs in a DiffServ router can monitor packet statistics of multiple priority queues for DiffServ traffic classes. The per-class statistics are basic data for performing edge-to-edge throughput monitoring.

The final assumption is related to drop characteristic of priority queues in DiffServ routers. The suggested edge-to-edge throughput monitoring methods need each priority queue to have the proportional drop characteristic. The proportional drop means that the number of packet drops at a queue is proportional to the amount of packets transmitted. The proportional drop is important because the suggested monitoring methods calculate the amount of packet drops of edge-to-edge DiffServ flows by using the characteristic. This assumption is proved valid by simulation in Section 4.5.

4.2 Modeling Edge-to-Edge DiffServ Flows

A DiffServ router can handle QoS information of multiple DiffServ classes. If a router r_j handles packets of DiffServ class of c_i, we can denote the QoS of the

packets of DiffServ class of c_i experienced in the router r_j, is $Q(c_i,r_j)$. QoS information can include various performance parameters, such as throughput, jitter, delay, number of packet drops, etc.

In order to construct an edge-to-edge model of DiffServ flows, it is necessary to have a topological model of DiffServ flows, as well as a QoS information model because edge-to-edge DiffServ flow is built on a routing path from one edge router to the other edge router. The topology of a DiffServ domain is denoted as a set of DiffServ routers (r_i) interconnected with unidirectional routing path. A unidirectional link (l_i) exists from router r_1 to r_2 if and only if there is a direct routing path from r_1 to r_2. Thus, a topology of a DiffServ domain, T can be denoted as follows.

$$T = R \cup L, \text{ where } R = \{ r_1, r_2, r_3, \ldots, r_m \}, \text{ a set of } m \text{ routers}$$
$$L = \{ l_1, l_2, l_3, \ldots, l_n \}, \text{ a set of } n \text{ links}$$

In addition to the topological modeling, the dynamic traffic information should be modeled as well. At a certain moment, there is a traffic flows between a set of edge routers within the DiffServ domain. This is denoted as a DiffServ flow. Since a DiffServ domain carries different classes of DiffServ traffic, a DiffServ flow of class c_i, $D(c_i)$ can be represented as follows.

$$D(c_i) = (T_i, Q_i), \text{ where } T_i = R_i \cup L_i \subseteq T$$
$$Q_i = \{ Q(c_i, r_k) \mid r_k \in R_i \}$$

T_i is a subset of topology T and contains a set of routers and links, where the DiffServ flow of class c_i is detected and monitored.

An edge-to-edge DiffServ flow of concern is a set of IP packets flowing from one edge router to another edge router with the same DSCP value in a certain period of time. Thus, the edge-to-edge DiffServ flow of class c_i, source edge router r_s, and destination edge router r_d can be denoted as $D(c_i, r_s, r_d)$ and

represented as the following equation.

$$D(c_i, r_s, r_d) = (T_{i,r_s,r_d}, Q_{i,r_s,r_d}), \text{ where } T_{i,r_s,r_d} = R_{i,r_s,r_d} \cup L_{i,r_s,r_d} \subseteq T_i$$

$$Q_{i,r_s,r_d} = \{ Q(c_i, r_k) \mid r_k \in R_{i,r_s,r_d} \}$$

T_{i,r_s,r_d} is a subset of DiffServ flow T_i and represents the routing path from source edge router, r_s, to destination edge router, r_d.

Figure 8 depicts mapping relationships of three hierarchical models. A topology model contains a set of DiffServ flows of different DiffServ classes, and a DiffServ flow model contains a set of different source/destination edge-to-edge DiffServ flows. The graphical representation reveals how to extract an edge-to-edge DiffServ flow from a topology of a DiffServ domain.

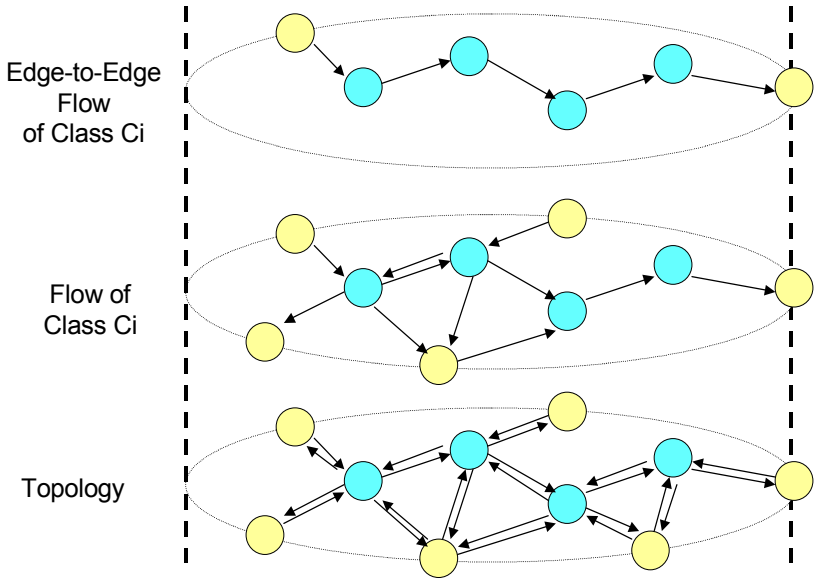


Figure 8. Mapping Relationships of Topology, Flow, and Edge-to-Edge Flow

When managing DiffServ networks, it is beneficial to possess information on edge-to-edge DiffServ flows in a DiffServ domain. First, edge-to-edge DiffServ flows can be easily mapped to various network services provided by a DiffServ

domain. Topology, status, and performance of the given network service is easily retrieved from the edge-to-edge DiffServ flows information. Thus, service management operations can be simplified and various high-level management operations, such as accounting and billing, SLA negotiation and monitoring, can be constructed on the edge-to-edge DiffServ flows. Secondly, routing decisions can be combined with traffic conditioning decisions. Without edge-to-edge DiffServ flows, routing is totally independent of traffic conditioning. This might improve the performance of routers, but the separation makes traffic conditioning ignorant of the routing topology. If traffic conditioning decisions can be made with the knowledge of routing or vice versa, better decisions would be made. The edge-to-edge DiffServ flows combine the routing and the traffic conditioning to assist both decisions to become more effective and efficient. Lastly, runtime dynamics of network traffic are represented in edge-to-edge DiffServ flows. Dynamically changing behaviors can be classified in the DiffServ flows, and administrators can easily understand and manipulate network traffic by managing DiffServ flows, not by managing individual routers.

When managing edge-to-edge QoS of a DiffServ flow, we can use two different approaches. One is measuring QoS at each edge point, and the other is measuring QoS in every routing point on routing path. Our approach is a distributed measurement of QoS information at every transit router between two edge routers. And the locally measured QoS information is aggregated by predefined rules.

The distributed QoS measurement and concatenation method is explained in Figure 9. Following the routing path from the source edge router, E_s, to the destination edge router E_d, there are n core routers from R_1 to R_n located serially. Each core router can monitor and collect QoS information of traffic flows and keep the information as Q_i. The distributed QoS observation from one router does not interact with other routers so that every transit router gathers QoS information independently. When the edge-to-edge QoS is needed, the locally-observed QoS are aggregated hop by hop.

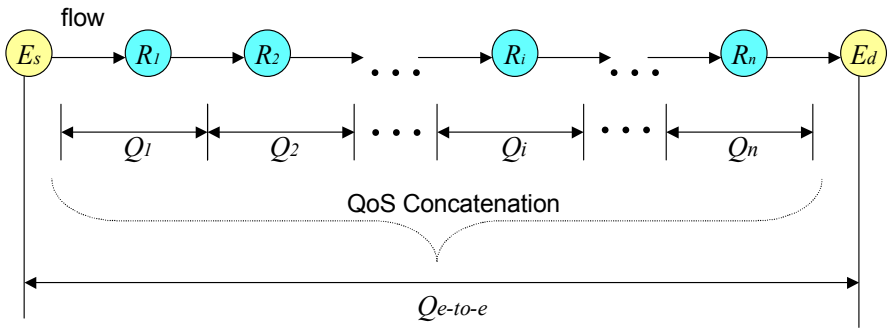


Figure 9. Distributed QoS Measurement and Concatenation

Information obtained from edge-to-edge DiffServ flows consists of two parts: topology and performance. Topology information represents router-to-router connectivity. A path from a set of source edge routers to a set of destination edge routers must be provided. Performance information represents a number of performance parameters of a given DiffServ path. The performance information can be obtained by combining performance parameters of each router in a DiffServ path.

To represent edge-to-edge DiffServ flows more efficiently, A set of graphical notation of topology is devised. Basic graphical notations of DiffServ flows are showed to understand the topological information of each DiffServ flow.

Figure 10 illustrates four different kinds of DiffServ node representations, composed of a vertex and directed edges.

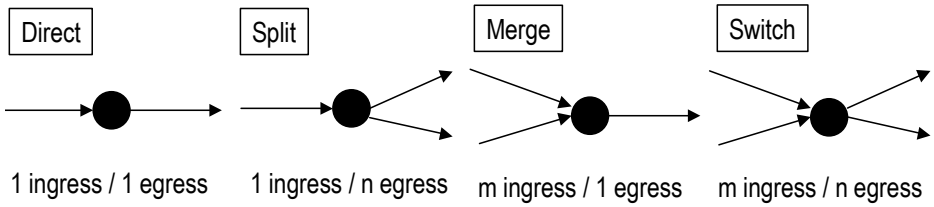


Figure 10. Graphical Representation of DiffServ Nodes

A direct node receives DiffServ flows from one direction and forwards it to only one direction. A split node receives DiffServ flows from one direction but forwards them to two or more directions. A merge node receives DiffServ flows from multiple directions and combines them to one outgoing direction. A switch node splits and merges at the same time from multiple incoming directions to multiple outgoing directions. The reason why there are different types of DiffServ nodes is that different kinds of network nodes require different parameter aggregation rules.

Without loss of generality, the switch node can be converted to a linkage of one merge node and one split node as in Figure 11. A virtual link with no propagation delay connects a merge node with the same ingress edges in the switch node and a split node with the same egress edges in the switch node. This conversion enables the topology of an edge-to-edge DiffServ flow to be represented without the switch node and makes the aggregation rules simpler. With the switch nodes, the aggregation rules need to handle more complicated cases.

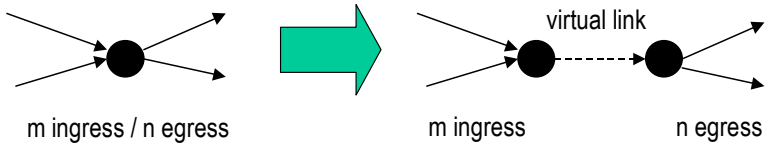


Figure 11. Conversion of a Switch Node

Thus, an edge-to-edge DiffServ flow can be represented as a linked list of DiffServ nodes, which are one of direct, merge, and split nodes. A link belonging to the edge-to-edge routing path is named ‘path link,’ a link belonging to a split node and out of the routing path is named ‘split link,’ and a link belonging to a merge node and coming in the routing path is named ‘merge link.’ At each hop of the routing path from the source edge router to the destination edge router, only one path link exists. Following the routing path, the split links extract traffic from the edge-to-edge path and the merge links add traffic to the edge-to-edge path. A

graphical representation of these links in an edge-to-edge DiffServ flow is illustrated in Figure 12.

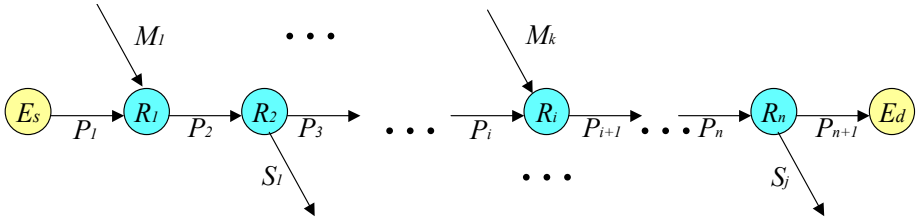


Figure 12. Graphical Representation of an Edge-to-Edge DiffServ Flow

The directed graph in Figure 12 is a flow network [20] with multiple sources and sinks in graph theory. If we assume that the flow network is in steady state within the monitoring interval, we can say that the amount of traffic coming in the flow network at the source path link (P_1) and the merge links (M_1 to M_k) in the routing path is the same as the amount of traffic going out from the flow network at the destination path link (P_{n+1}) and the split paths (S_1 to S_j) in the routing path. Even if the packet drops occur due to congestion in some of the links, the equilibrium property still holds true.

4.3 Edge-to-Edge Throughput with Min/Max Bound

From the modeling of an edge-to-edge DiffServ flow, a traffic flow of a DiffServ class from one edge router to another edge router within the monitoring interval can be graphically represented with an array of DiffServ routers and a set of network links. Since each DiffServ router monitors QoS statistics for the aggregated traffic of a DiffServ class, not for the edge-to-edge DiffServ flow of concern, it is needed to devise how to construct the edge-to-edge QoS statistics from the statistics for aggregated traffic. Locally-monitored performance information and the edge-to-edge routing path are used to generate edge-to-edge QoS of DiffServ flows. Simple hop-by-hop QoS concatenation rules are proposed

for concatenating QoS values of parameters monitored at each DiffServ node in edge-to-edge routing path. The purpose of the concatenation rules is to extract status of an edge-to-edge DiffServ flow from locally-observed traffic statistics.

The first approach is to find the minimum and maximum amount of throughput of the edge-to-edge DiffServ flow from the locally-observed traffic statistics.

The maximum amount of throughput of the edge-to-edge DiffServ flow can be obtained by finding the minimum value among the throughput of all path links. Since the edge-to-edge throughput of a DiffServ flow is bounded by the link of the minimum throughput, the edge-to-edge DiffServ throughput is at most the value of the minimum throughput. Hop-by-hop concatenation rule is to find the smaller value between the ingress throughput and the egress throughput at each DiffServ router.

The minimum amount of throughput of the edge-to-edge DiffServ flow can be obtained by extracting the amount of throughput at split paths from the current edge-to-edge throughput value. The minimum throughput situation occurs when the initial traffic from the source edge router is divided at each split path as much as possible. Hop-by-hop concatenation rule is to extract the amount of throughput of the split path from the current minimum throughput value whenever encountering a split path within the edge-to-edge DiffServ model. At the direct and merge nodes, the same operation for calculating the maximum amount of throughput is applied.

With the above QoS concatenation rules, the maximum and the minimum amount of throughput of the edge-to-edge DiffServ flow is calculated. When the edge-to-edge throughput is found, we can see how much portion of the aggregated traffic at each DiffServ path link is consumed by the edge-to-edge DiffServ flow. Without the edge-to-edge DiffServ concepts, network administrators cannot tell the portion for each internal flow. This bandwidth sharing within the aggregated traffic can be quantified by bandwidth sharing ratio (BSR), defined as follows.

$$\text{bandwidth sharing ratio (BSR)} = \frac{\text{amount of edge-to-edge throughput}}{\text{amount of aggregated throughput of a link}}$$

The BSR is important for extracting the edge-to-edge DiffServ flow from mixed DiffServ flows in a link. If there are multiple ingress links and the DiffServ node does not distinguish different edge-to-edge DiffServ flows because of the same DSCP value, every egress link might have a mix of different edge-to-edge DiffServ flows. However, since we know the current amount of throughput of the edge-to-edge DiffServ flow in the ingress link and the DiffServ node treats packets the same way at the egress link, we can assume that performance parameters, such as throughput and number of dropped packets in the egress link, are divided according to the fraction of the amount of ingress link over the amount of egress link. For example, if the throughput of edge-to-edge DiffServ flow of concern is 10 Mbps in the ingress link, while the throughput of the next hop egress link is 20 Mbps, we assume that a half of the egress link traffic comes from other edge-to-edge DiffServ flows. In this situation, the BSR is 0.5, and so the throughput and number of dropped packets of the egress link should be multiplied by the fraction to extract parameters of edge-to-edge DiffServ flow of concern.

4.3.1 Min/Max Throughput Monitoring Algorithms

The edge-to-edge throughput monitoring has two phases. The first phase is for calculating edge-to-edge throughput following routing path in forward direction and the second phase is for calculating edge-to-edge drop following routing path in reverse direction. The second phase uses BSR of the edge-to-edge DiffServ flow at each link in the routing path calculated in the first phase.

The reason why two separate phases are needed is that a single phase cannot calculate exact amount of packet drops for the edge-to-edge DiffServ flow because the final edge-to-edge throughput is not calculated while progressing the first phase. Thus, after the final edge-to-edge throughput is calculated, the second

phase can calculate the exact amount of packet drops at each link by using BSR of the edge-to-edge DiffServ flow. The first and second phase algorithms are described as follows.

Min/Max Throughput, First Phase

Input: throughput at each link, edge-to-edge topology
Output: minimum and maximum of final edge-to-edge throughput

```
TH_max = TH_min = throughput of P1

for ( each path link, Pi ) {
    BSR_max = TH_max / throughput of all merged links
    DROP_max = BSR_max * Di
    TH_max = TH_max - DROP_max

    BSR_min = TH_min / throughput of all merged links
    DROP_min = BSR_min * Di
    TH_min = TH_min - DROP_max

    TH_max = min( TH_max, throughput of Pi+1 )
    if ( Ri is a split node ) {
        TH_min = TH_min - throughput of all split links
    } else {
        TH_min = min( TH_min, throughput of Pi+1 )
    }
}

final_throughput_max = TH_max
final_throughput_min = TH_min
```

Min/Max Throughput, Second Phase

Input: throughput and drop at each link, edge-to-edge topology

Output: minimum and maximum of initial edge-to-edge throughput, transit drop at each link

```
TH_max = final_throughput_max
TH_min = final_throughput_min

for ( each path link, Pi, in reverse order ) {
    BSR_max = TH_max / throughput of Pi
    DROP_max = BSR_max * Di
    TH_max = TH_max + DROP_max

    BSR_min = TH_min / throughput of Pi
    DROP_min = BSR_min * Di
    TH_min = TH_min + DROP_min
}

initial_throughput_max = TH_max
initial_throughput_min = TH_min
```

4.3.2 Min/Max Throughput Monitoring Examples

Two detailed examples (one with no packet drop and one with packet drops) are illustrated with the routing topology in Figure 13. An edge-to-edge DiffServ flow starts at a source edge router E_s and sinks at a destination edge router E_d.

There are four core routers, labeled from R_1 to R_4 with path links from P_1 to P_5, merge links M_1 and M_2, and split links S_1 and S_2. Thus, routers R_1 and R_3 are merge nodes and routers R_2 and R_4 are split nodes.

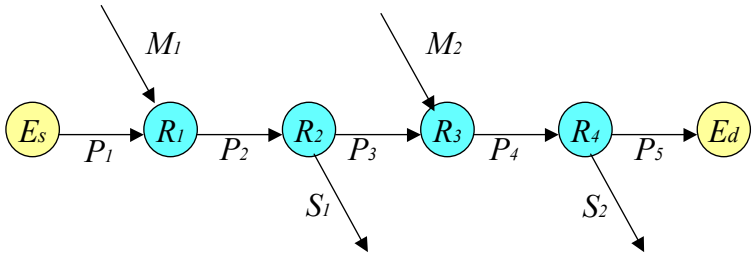
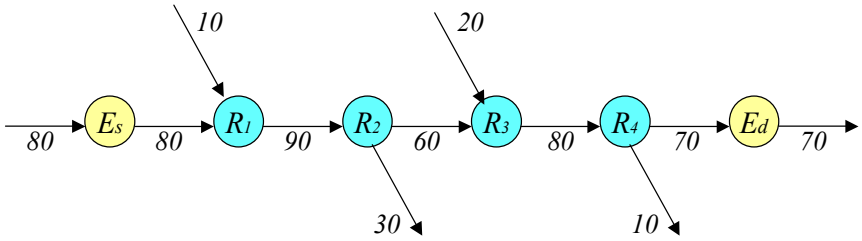


Figure 13. Example Topology of an Edge-to-Edge DiffServ Flow

The first example case is in a situation where all links have no packet drops. There is no congestion point in this situation and packets injected to the network are completely transferred to the desired destination node. Example QoS monitoring information and edge-to-edge concatenation in this situation is given in Figure 14. The number at each link represents a relative value of the measured bandwidth within a monitoring interval at the link.



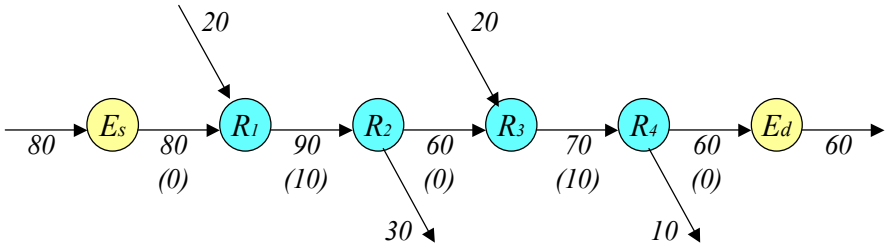
Min Throughput	80	80	50	50	40	final minimum
Min BSR		40/90	40/60	40/80	40/70	
Max Throughput	80	80	60	60	60	final maximum
Max BSR		60/90	60/60	60/80	60/70	

Figure 14. Example of Edge-to-Edge Concatenation Rules (No Drop Situation)

At first, the initial throughput received by the source edge router is 80. By following the routing path and the QoS concatenation rules, it is found that the maximum edge-to-edge throughput of the DiffServ flow is bounded to the value of 60 at path link P_3. And the minimum edge-to-edge throughput of the DiffServ flow is bounded to the value of 40 because there are two split paths, S_1 and S_2, with the divided throughput of 30 and 10 respectively. The amount of the divided throughput is extracted from the amount of initial throughput. At each network link, the BSR value is calculated to represent how much outgoing bandwidth is reserved for the edge-to-edge DiffServ flow.

The second example illustrated in Figure 15 and Figure 16 is in a situation where there are congestion points where available bandwidth is less than the traffic demand at some network links and thus packets are dropped at the bottleneck links. The numbers within parentheses represent the relative amount of packet drops monitored at each link. The sum of throughput and packet drops at an outgoing link is the same as the total throughput at the previous incoming links.

The min/max throughput calculation is basically the same as the previous example. However, since there exist packet drops, we need to calculate how much portion of all packet drops measured at the bottleneck link for the aggregated traffic is related to the edge-to-edge DiffServ flow. The QoS concatenation is performed in two phases. The first phase is for calculating minimum and maximum throughput for the edge-to-edge DiffServ flow, and the second phase is for calculating the number of packet drops of the edge-to-edge DiffServ flow because the number of packet drops calculated in the first phase includes packet drops of split traffic flow that should be eliminated for the edge-to-edge DiffServ flow. The split traffic flow starts from the same ingress router, but ends to the different egress router. The first phase example is illustrated in Figure 15 and the second phase example is in Figure 16.



Min Throughput	80	72	42	37	27	final minimum
Min BSR		80/100		42/80		
Drop at Link	0	4/5 × 10	0	42/80 × 10	0	
Total Drop for Min	0	8	8	13	13	

Max Throughput	80	72	60	52	52	final maximum
Max BSR		80/100		60/80		
Drop at Link	0	4/5 × 10	0	6/8 × 10	0	
Total Drop for Max	0	8	8	16	16	

Figure 15. Edge-to-Edge Concatenation (Drop Situation, First Phase)

There are two bottleneck links, P_2 and P_4, in this example. At each bottleneck link, the BSR value is determined to calculate the portion of packet drops for the edge-to-edge DiffServ flow. The overall calculation for the minimum throughput is performed as follows.

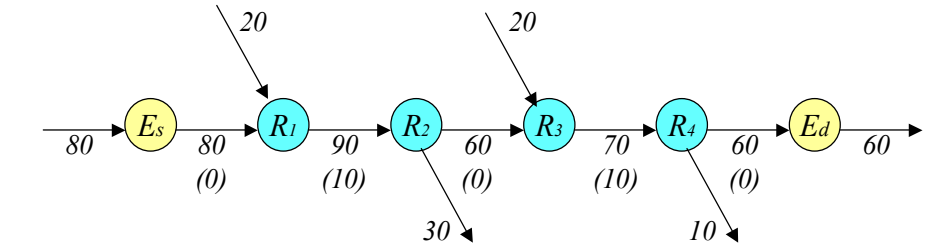
The initial amount of traffic received by the ingress edge router is 80. The first node, R_1, is a merge node and receives merged traffic of 20, which is added to the current throughput of 80. Since R_1 is a merge node, the BSR value is calculated as 80/100 because the merged total throughput is 100 and the received throughput from the routing path is 80. Now the merged traffic is also shaped with the drop of

10 at the path link P_2 and thus the amount of throughput transferred to R_2 shrinks to 80. When calculating number of dropped packets, the BSR value is used. Since, at P_2, the merged traffic is treated with the same shaping processes, we can assume the drop probability is evenly distributed over all incoming packets. Thus, among the amount of dropped packets of 10, only the BSR portion of it affects the edge-to-edge DiffServ flow of concern. This is calculated by multiplying the BSR value to the current amount of dropped packets and resulted in value of 8. Thus the current edge-to-edge throughput shrinks to 72 by extracting the amount of dropped packets. At the node R_3, since this node is a split node with split traffic volume of 30, the minimum throughput shrinks to 42 by extracting the amount of split traffic. At the path link P4, the same BSR calculation is performed and the results amount of packet drops is 5. The total amount of dropped packets in the edge-to-edge DiffServ flow is accumulated to 13 at path P_4. Finally, by calculating split path of S_2, the final minimum throughput of the edge-to-edge DiffServ flow becomes 27.

The maximum throughput is calculated by following the same steps in the first example and the final value is 52. The only difference is that for the maximum throughput, the split path is not considered and the smaller value between incoming and outgoing throughput of each path link is selected.

In order to calculate the amount of packet drops for the edge-to-edge DiffServ flow, we need to apply similar concatenation rules again. In the second phase, the calculation starts from the egress router, which is in reverse order. The reason why we need the second phase for the edge-to-edge drops is that the amount of dropped packets from the first phase does not represent the actual amount of dropped packets of the edge-to-edge DiffServ flow of concern. We need to re-calculate it from the final minimum and the final maximum value. The second phase example is described in Figure 16.

At the egress point, there is the final minimum and the final maximum throughput values from the first phase calculation. From the egress router to the ingress router in reverse order, the hop-by-hop calculation is as follows.



initial minimum

Min Throughput	34	31	31	27	27	final minimum
Min BSR		31/90		27/70		
Drop at Link	0	31/90 × 10	0	27/70 × 10	0	
Total Drop for Min	7	7	4	4	0	

initial maximum

Max Throughput	66	59	59	52	52	final maximum
Max BSR		59/90		52/70		
Drop at Link	0	59/90 × 10	0	52/70 × 10	0	
Total Drop for Max	14	14	7	7	0	

Figure 16. Edge-to-Edge Concatenation (Drop Situation, Second Phase)

At the path link P_4, the total amount of dropped packets is 10. However, since the BSR of the edge-to-edge DiffServ flow is below 1, only the portion of the amount of dropped packets is related to the edge-to-edge DiffServ flow. For the minimum throughput, the BSR is 27/70 and the calculated amount of dropped packets is 4. For the maximum throughput, the BSR is 52/70 and the calculated amount of dropped packets is 7. Then, before moving forward to the previous router, we need to add the amount of dropped packets to the current edge-to-edge throughput value because the amount of dropped packets had been transferred before the bottleneck link. Thus, the total throughput increases to 31 for the

minimum, and to 59 for the maximum. The same calculation is applied at the path link P_2, and the final amount of dropped packets for the maximum throughput case is 7 while the value is 14 for the minimum throughput case.

With the two-phased calculations, we can determine the maximum and the minimum value of throughput of the edge-to-edge DiffServ flow from the locally-observed statistics for the aggregated traffic. When packets are dropped in some of routers in the routing path, we can calculate how much portion of packet drops occurs for the edge-to-edge DiffServ flow. With this drop information, the amount of throughput initially injected at the ingress router is also projected.

4.4 Edge-to-Edge Throughput with Ingress Marking and Egress Counting

In order to monitor edge-to-edge QoS of a DiffServ flow, it is essential to know the amount of traffic between two specific source and destination edge routers. This section suggests a mechanism to measure the amount of traffic between every source/destination pair by using an ingress marking / egress counting (IM/EC) method. The overall architecture of the proposed method is described in Figure 17.

At every edge router, the edge router marks its identifier in packet header and this information reaches the destination edge router. When an edge router receives a packet, the router can distinguish which edge router injects this packet into the DiffServ domain. With this mechanism, the every edge router keeps the amount of traffic from every other edge router. This information is referred as ‘E-to-E throughput matrix’ hereafter.

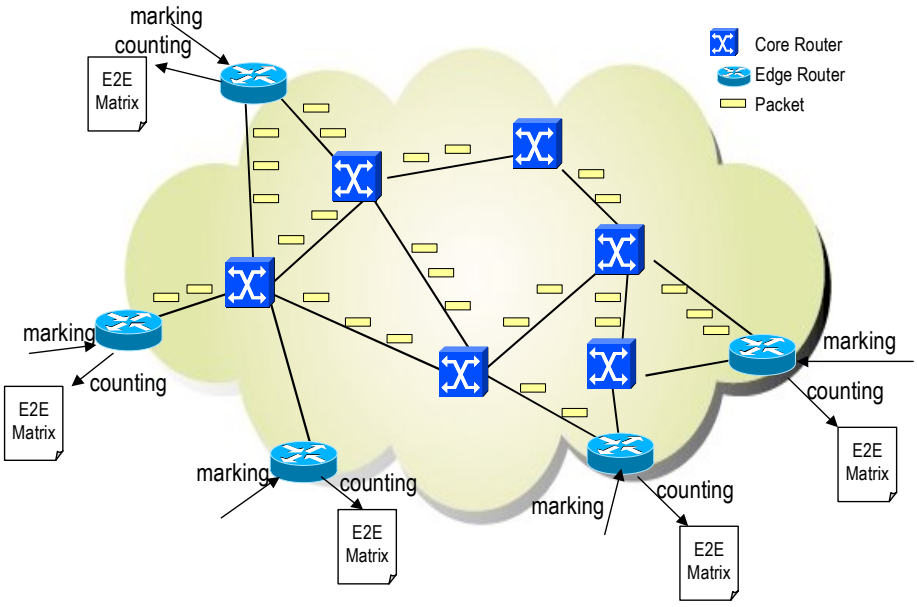


Figure 17. Source Marking / Destination Counting Architecture

4.4.1 IM/EC Throughput Monitoring Algorithms

With the E-to-E throughput matrix, the first phase algorithm described in the previous min/max monitoring method is not necessary. When an edge-to-edge DiffServ flow is specified with a combination of ingress and egress edge routers, the only thing to calculate the edge-to-edge throughput of the DiffServ flow is looking up the E-to-E throughput matrix stored in the egress router and searching for ingress edge router ID in the matrix. Thus, only the second phase algorithm is needed. The second phase algorithm is basically the same as in the min/max monitoring method. While following the routing path in the reverse order, the second phase algorithm calculates BSR of the edge-to-edge DiffServ flow and amount of packet drops at each link for the edge-to-edge DiffServ flow. And then, the second phase algorithm increases current throughput with the amount of packet drops so that the initial injected throughput at the ingress edge router is

calculated in the end.

IM/EC Throughput, Second Phase

Input: throughput and drop at each link, edge-to-edge topology
Output: initial edge-to-edge throughput, transit drop at each link

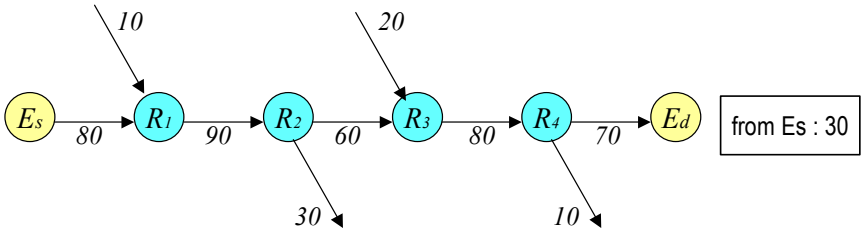
```
TH = edge-to-edge throughput from the E-to-E matrix

for ( each path link, Pi, in reverse order ) {
    BSR = TH / throughput of Pi
    DROP = BSR * Di
    TH = TH + DROP
}

initial_throughput = TH
```

4.4.2 IM/EC Throughput Monitoring Examples

With the local QoS monitoring information and the E-to-E throughput matrix at edge routers, the edge-to-edge DiffServ QoS can be obtained as the following two examples. One example is in a situation without packet drop (Figure 18) and the other example is with packet drops at transit routers (Figure 19).

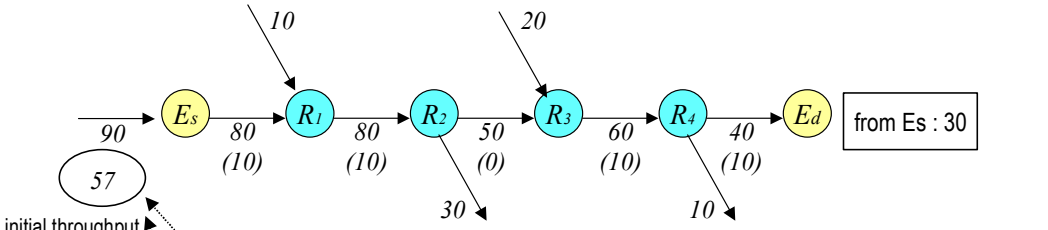
When there is no packet loss across the whole routing path, which is unloaded network situation, the amount of traffic marked from ingress *Es* is the same as the amount of traffic leaving *Es*. By dividing the amount by the total bandwidth utilization at each link, the BSR of the edge-to-edge traffic flow at the link is obtained. For example as in Figure 18, at the first link, the BSR is 3/8 and it

becomes 1/3 at the next link. With this BSR value, network administrators can find how much portion of one specific link's utilization is consumed by the edge-to-edge traffic flow of concern.



Throughput	30	30	30	30	30
Share Ratio	30/80	30/90	30/60	30/80	30/70

Figure 18. Example QoS Concatenation (No Drop Situation)



Throughput	50	44	44	38	30
Share Ratio	64/80	44/80	44/50	38/60	30/40
Drop at Link	7	6	0	6	8
Total Drop	27	20	14	14	8

round off value

Figure 19. Example QoS Concatenation (Drop Situation)

When network resources become scarce and there are multiple bottleneck points between two edge routers as in Figure 19, the total amount of traffic left

from ingress router cannot reach egress router. At bottleneck links, some portion of the traffic should be discarded. At the egress router, the counted amount of traffic from a certain ingress router *Es* becomes much less than the amount of original traffic left *Es*. In this situation, it is important to know where the packets are dropped and how much portion of the edge-to-edge traffic flow is discarded at the link.

As depicted in Figure 19, if the amount of traffic from ingress router *Es* is counted to 30 at the egress router Ed, the amount of dropped packets at each routing path can be calculated as follows. The calculation starts from the path link adjacent to egress router and the calculation ripples to the ingress router link by link. There two kinds of numbers at each link, the number without parentheses means the amount of traffic measured at each link and the number within parentheses means the amount of traffic dropped at each link. These values measured in core routers are for aggregated flow, which means the total amount of traffic belonging to a DiffServ class, not for each edge-to-edge flow. Thus, the amount of edge-to-edge flows from ingress *Es* and egress *Ed* is included in the measured value.

At first, at the last path link adjacent to *Ed*, the bandwidth sharing ratio of the edge-to-edge flow is calculated. Since the amount of marked traffic measured at *Ed* is 30 and the amount of total traffic measured at the path link is 40, the BSR becomes 3/4. Next, the amount of dropped packets is calculated by multiplying the BSR to the total amount of dropped traffic at the path link. In order for this calculation to be correct, the drop probability at each link should be independent of number of edge-to-edge flows. That is, the likelihood of dropping packets should be fair and proportional to the amount of traffic flowing through the link. This assumption will be proved true when the link uses RED-typed queuing disciplines in the next section. Thus, the amount of dropped traffic for the edge-to-edge DiffServ flow is calculated to 8 among total amount of dropped traffic of 10. Now, the total amount of edge-to-edge traffic reached at the path link is assumed to be 38 by adding the amount of dropped traffic at the path link. Applying the

same calculation rule at each path link one by one, the final throughput left ingress router *Es* is calculated to 57 among total amount of traffic left of 90.

4.5 Proportional Drop Simulation and Verification

The proposed method assumes that drop probability of an edge-to-edge DiffServ flow is proportional to the amount of bandwidth shared by the flow at the link. It is required that this property is actually valid in simulation environment. DiffServ routers use Random Early Detection (RED) [66] queues to drop excessive amount of packets when congestion occurs. In this section, the proportional drop rate is investigated in a simulation network by using ns-2 discrete event simulator.

RED uses the queue's average occupancy as a parameter to a random function that decides whether congestion-avoidance mechanism ought to be triggered. As the average occupancy increases, the probability of a packet-drop action increases. Figure 20 shows a typical drop probability function of RED queue.

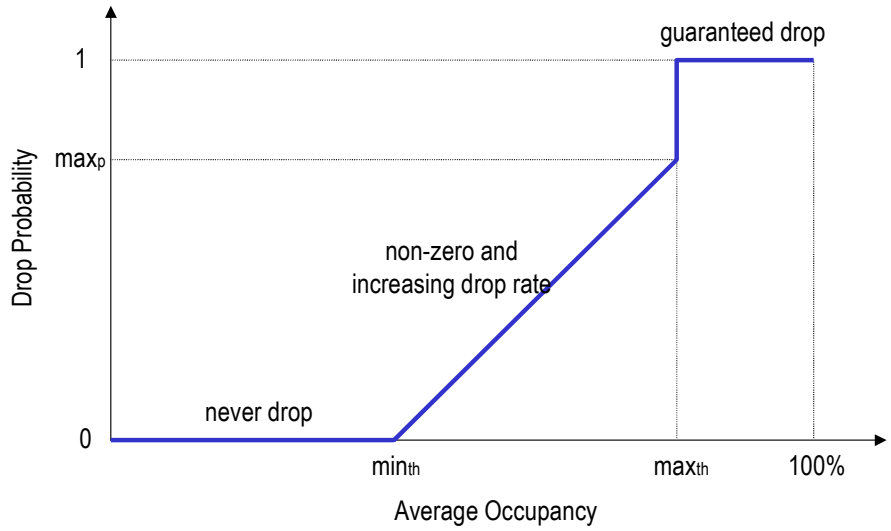


Figure 20. RED Drop Probability Function

For occupancy up to a lower threshold, min_th, packets pass through untouched. Above min_th, the probability of packet drop rises linearly toward a probability of max_p reached for an occupancy of max_th. At and above max_th, packets are guaranteed to be dropped.

These three phases are sometimes referred to as normal, congestion avoidance, and congestion control, respectively. The worst-case long-term queue size is limited to max_th. RED begins early in that it begins triggering congestion indications well before the queue becomes full. RED discards packets in proportion to the bandwidth used by the flows. This means that edge-to-edge flows competing for bandwidth allocated to an RED queue experience relative drop rate according to each flow's BSR.

To verify the relative drop rate of edge-to-edge DiffServ flows, a simple merge node is modeled with ns-2 network simulator. Figure 21 describes the modeling of a merge node with RED queue.

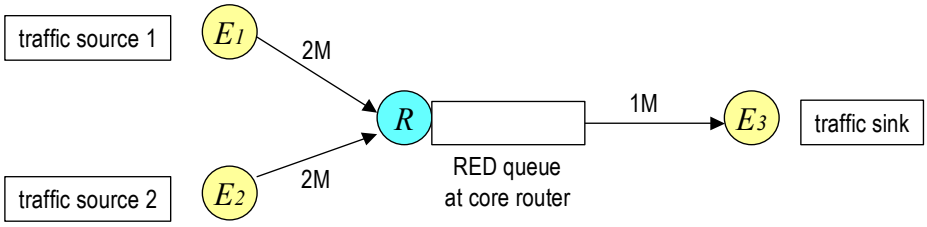


Figure 21. Modeling an RED Merge Node

There are three edge routers (E1, E2, and E3) and one core router (R) in the modeling. The edge routers, E1 and E2 are ingress edge routers with traffic sources, and the edge, E3, is an egress edge router with a traffic sink. Thus, there are two kinds of traffic flows from E1 to E3, and from E2 to E3. The bandwidth allocation for this DiffServ class in each link is represented on each path. 2Mbps is allocated for E1-R and E2-R links and 1Mbps is for R-E3 link. Within core router, R, an RED queue merges two different traffic flows (one from E1 and the other from E2) and discards packets when congestions occur.

In order to make the R-E3 link saturated and to investigate the proportional drop rates, the total traffic arriving at router R is configured to be maximum of 1.2Mbps, which is 20% over the maximum allocated bandwidth of 1Mbps. The bandwidth shares of two edge-to-edge DiffServ flows are differently configured at each simulation step. The bandwidth share starts 1 to 9, which means that the amount of traffic from E2 is 9 times more than the amount of traffic from E1. And then the amount of traffic from E2 is gradually increased and the amount of traffic from E1 is gradually decreased. Finally, the bandwidth share becomes 9 to 1. For each simulation step, the amount of dropped packets for each flow is recorded. Table 3 summarizes the simulation results.

Table 3. Proportional Drop Simulation Result

BSR	E1→E3	E2→E3	Total RED Drop	Drop from E1	Drop from E2
1:9	0.12Mbps	1.08Mbps	3238	325 (10.04%)	2913 (89.06%)
2:8	0.24Mbps	0.96Mbps	3219	649 (20.16%)	2570 (79.84%)
3:7	0.36Mbps	0.84Mbps	3246	956 (29.45%)	2290 (70.55%)
4:6	0.48Mbps	0.72Mbps	3219	1246 (38.71%)	1973 (61.29%)
5:5	0.6Mbps	0.6Mbps	3267	1658 (50.75%)	1609 (49.25%)
6:4	0.72Mbps	0.48Mbps	3229	1944 (60.20%)	1285 (39.80%)
7:3	0.84Mbps	0.36Mbps	3227	2271 (70.37%)	956 (29.63%)
8:2	0.96Mbps	0.24Mbps	3210	2581 (80.40%)	629 (19.60%)
9:1	1.08Mbps	0.12Mbps	3391	3065 (90.39%)	326 (9.61%)

The simulation results show that the RED queue drops packets in proportion to the amount of bandwidth share of each merged traffic flow. Although there exist small errors in each simulation, the RED queue dropped packets with the same ratio over two different DiffServ flows. Figure 22 depicts the relationship in graph. The projected line represents the initial bandwidth share of traffic sources

and the red dots in the graph represent the simulated drop ratio of DiffServ flows. Almost all simulations show that the RED drops packets in proportion to the amount of bandwidth shares in its input link.

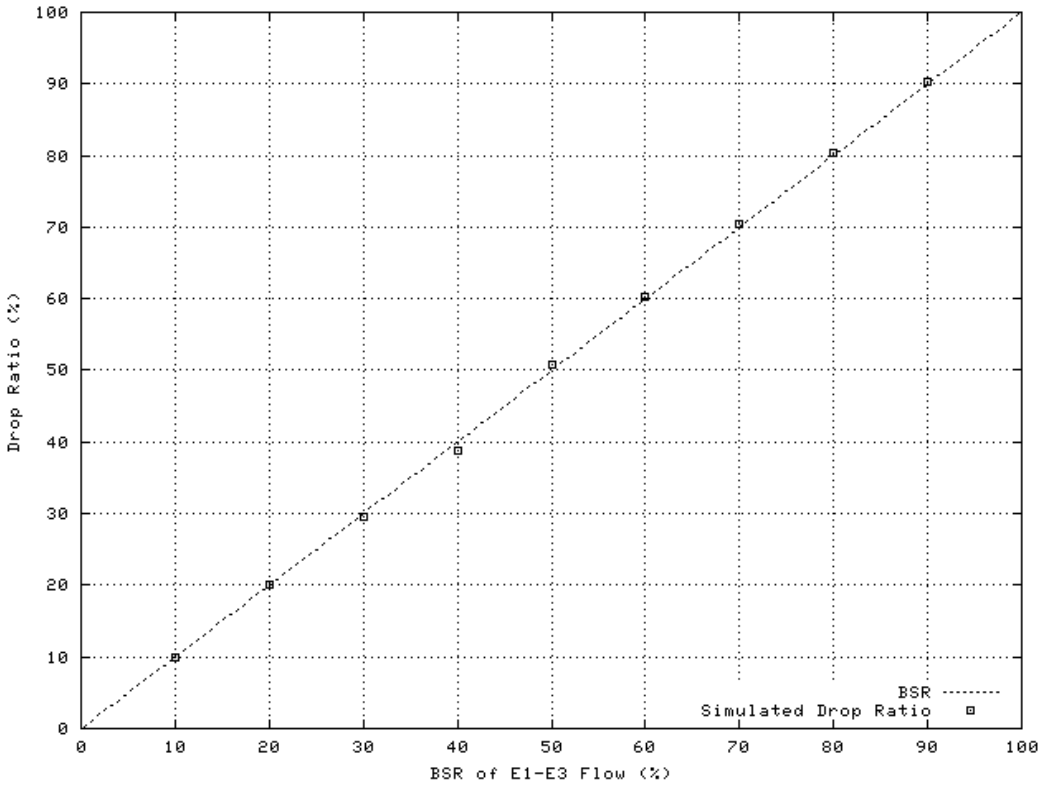


Figure 22. BSR vs. Simulated Drop Ratio

With the above simulation results, it is concluded that RED queues used in DiffServ routers actually drop packets in proportion to the bandwidth share of each DiffServ flows. In addition, since each DiffServ router independently queues and processes packets, the edge-to-edge drop ratio calculated by concatenating drop ratio at each RED queue is valid. Despite a limited number of errors between calculation and actual packet drops due to the dynamic nature of communication network, the errors are not correlated to each other and can be ignored when calculating edge-to-edge drop ratio.

4.6 Implementation Considerations

When implementing the proposed method in DiffServ networks, it is needed to consider the following considerations. The first consideration is about the detailed implementation of source edge marking. The second consideration is about monitoring interval when collecting statistical information from each DiffServ router.

4.6.1 Source Marking Methods

The proposed ingress marking / egress counting method needs additional marking in IP packet header. This requires additional header space for bit marking for edge routers. Here are two proposals for implementing additional header mark.

A straightforward marking method uses the existing DSCP field. The ToS byte containing DSCP field is 8-bit long (6 bits are for DSCP). If we reconfigure the structure of ToS byte to contain both DSCP and edge ID, the ToS byte can be used for source marking method. For example, the 8-bit ToS byte can be divided to two regions, one for DSCP and the other for edge ID. 4-bit DSCP can contain 16 different codes and 4-bit edge ID can contain 16 different IDs. Thus, if core routers are configured to check only the modified DSCP bits and edge routers are configured to mark edge ID at edge ID field, the DiffServ network can carry source-marked IP packets in its domain and count the amount of traffic between each edge routers.

The method using the modified DSCP field has limitations in providing enough numbers of edge routers and DSCP because the method packs two different information only in one byte (8 bits). The 8-bit limitation can be affordable in some small DiffServ networks with a few service classes and a few edge routers. However, in more complicated DiffServ domains, the modified DiffServ method cannot be applied. For this generalized environment, additional

header field should be used. Current IP packet header does not have additional space for this purpose. The possible mechanism is using MPLS scheme. Since MPLS has 20-bit shim header and the header can be stacked multiple times, theoretically there is no limitation to contain additional information in IP packet header. Various kinds of methods for embedding edge ID in shim header can be organized. The ToS byte in the original IP packet header is not modified at all in MPLS method.

Each of the two methods has pros and cons. The modified DSCP method can be applied without changing existing DiffServ network devices. The only thing to change is dividing the ToS byte into two fields. The core and edge routers are not modified either. The edge routers need to mark both edge ID and DSCP and core routers need to look up the modified DSCP values. However, overall modifications are only limited to configuration parameters. No hardware modification is needed. Thus, the modified DSCP method can be used in a small DiffServ network in the migration stage.

The MPLS method needs every DiffServ router to support the MPLS mechanism. The DiffServ routers in the domain should decode the MPLS header and forward the header to the next hop router. Every DiffServ router should additional change in packet handling, mostly in hardware modules. However, the MPLS-based DiffServ network is widely accepted as a future IP QoS network, it is likely that the MPLS network will be widely deployed in backbone networks. In this network environment, it is easy to add edge ID on MPLS header because each DiffServ router is already MPLS-ready. Also, the MPLS-based method does not have limited space for edge ID and DSCP value and thus general solution to big backbone networks.

4.6.2 Considerations on Monitoring Interval

When collecting statistical information from every DiffServ router, it is

difficult to define proper monitoring interval. If the monitoring interval is too short, system load for monitoring increases and statistical errors in collected data increases. However, if the monitoring interval is too wide, the short-term fluctuation in the statistical data is averaged out and not detected. Thus, proper monitoring interval is hard to decide. The actual monitoring interval in a DiffServ management system should be decided based on careful experience in the current system environment. The amount of information and the correctness of information is derivatively determined by the monitoring interval.

Two rules of thumbs are suggested for determining monitoring interval. One rule is that the monitoring interval should be longer than the longest edge-to-edge transmission delay. If the monitoring interval is shorter than the transmission delay, the collected information will omit the overall transmission characteristics of each flow. The monitoring interval should monitor complete transmission of each packet flow. The other rule is that the monitoring interval is shorter than congestion period. If a congestion period is shorter than the monitoring interval, some of edge-to-edge DiffServ flows detected within the period do not experience the congestion. Further, this might cause errors in calculating drop rates of each edge-to-edge DiffServ flows.

One more consideration in implementing DiffServ monitoring system is clock synchronization in each monitoring device. However, in a polling system, the clock synchronization is not important because polling system collects data in central server and central server can assume the information collected at a certain time period is measured in the same interval in each device.

4.7 Comparison of Two Monitoring Methods

The two monitoring methods for extracting edge-to-edge QoS statistics from aggregated local QoS monitoring information have pros and cons to each other. The first method (edge-to-edge QoS monitoring with min/max bound) extracts

throughput bound with maximum and minimum values without changing the current DiffServ framework. The second method (edge-to-edge QoS monitoring with ingress marking / egress counting) counts the exact amount of edge-to-edge DiffServ flow traffic with internal modifications on edge routers. Both methods calculates the amount of dropped packets of each edge-to-edge DiffServ flow by using the proportional drop characteristic of DiffServ RED queue and the amount of measured throughput of each edge-to-edge DiffServ flow.

In terms of accuracy of monitoring, the second method is more accurate than the first method. Since the first method only projects the deterministic bound for the actual edge-to-edge throughput, the gap between the minimum and the maximum bound can be large in some situations. Though the bound can be more useful for managing edge-to-edge DiffServ flows than the total aggregated statistics, the exact projection is needed in many service functionalities. The second method produces the exact amount of traffic between two different edge routers within a certain monitoring interval.

In terms of implementation easiness, the first method is easier to implement in the current DiffServ framework than the second method. The first method can be realized without any modification on the current DiffServ framework. The local QoS monitoring is the only thing needed for the method. However, the second method needs internal modifications on edge routers. Ingress edge routers are needed to mark edge ID on packets they inject to the DiffServ domain, and the egress edge routers are needed to check the edge ID and create edge-to-edge throughput matrix at each monitoring interval. Although this requires additional overhead on edge routers that handle relatively small amount of traffic, the method does not require any modification on core routers that handle huge amount of traffic in DiffServ networks. Thus, careful design and implementation of the second method may provide more feasible and accurate monitoring results.

5. Applications of the Proposed Monitoring Methods

In this chapter, the proposed monitoring methods from the previous chapter are applied to several management functions for DiffServ networks. The example scenarios in this chapter show the effectiveness and efficiency of the proposed monitoring methods.

5.1 Edge-to-Edge QoS Status Monitoring

Network administrators clearly need to have network status information that is dynamically changing due to numerous reasons in networks. With the proposed edge-to-edge monitoring method, the network administrators are able to collect network status efficiently. For every edge-to-edge pair, the amount of network traffic between the two edge routers is always counted within monitoring interval. With this QoS monitoring information, the network administrators easily construct whole network status and use the information to make QoS reconfiguration decisions.

With local monitoring information only, it is difficult to understand overall network status. However, with the proposed edge-to-edge QoS monitoring methods, the amount of traffics from one edge router to another edge router is reported and analyzed. For each network link, we can investigate how many edge-to-edge DiffServ flows are merged and compete for the reserved bandwidth and how much portion of the total bandwidth is shared by each edge-to-edge DiffServ flow. This information is important when congestion occurs and network administrators want to resolve bottleneck problems.

In order to show the effectiveness of edge-to-edge DiffServ QoS monitoring, a discrete simulation on a simple DiffServ network model has been performed. Figure 23 shows the simulation network with six DiffServ routers. Two routers

(S1 and S2) are ingress edge routers and two other routers (D1 and D2) are egress edge routers. Two routers (R1 and R2) are core routers in this DiffServ domain.

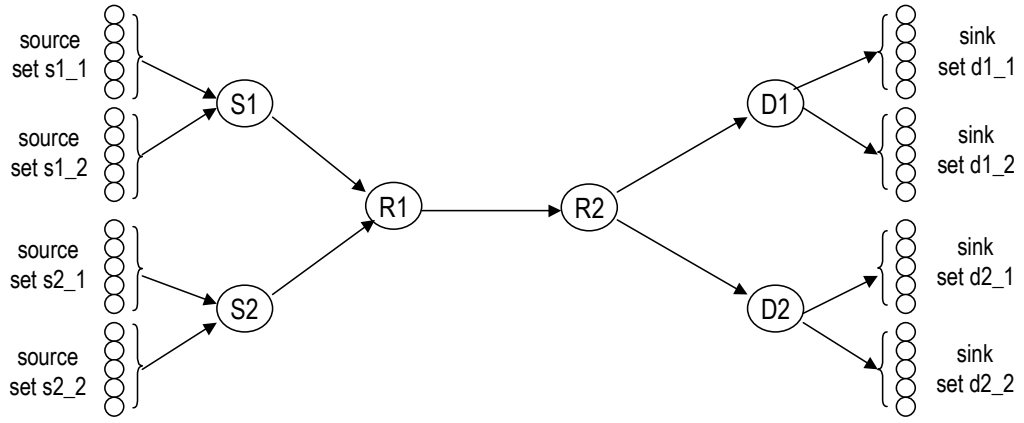


Figure 23. Simple Simulation Model

In each ingress edge router, there are 10 CBR traffic sources categorized in two source sets. Traffic sources within the same source set have the same destinations. In each egress edge router, there are 10 traffic sinks, also categorized in two sink sets. In order to make four different DiffServ flows, the traffic source set S11 is connected to the sink set D11 and the traffic source set S12 is connected to the sink set D21. Likewise, the traffic source set S21 is connected to the sink set D12 and the traffic source set S22 is connected to the sink set D22. Thus, four different DiffServ flows with different routing paths (S1 → D1, S1 → D2, S2 → D1, and S2 → D2) are constructed.

Each microflow connection randomly starts and stops generating CBR traffic during random amount of time. The minimum traffic generation rate is 64 bps and the maximum rate is 1 Mbps. For the simplicity, the link bandwidth of all network links is enough to carry all traffic without packet loss. During the simulation period of 20 seconds, the amount of aggregated DiffServ traffic measured in the link between R1 and R2 changes as in Figure 24. Ten randomly different microflows with different source and destination are aggregated in one DiffServ

class. With this statistics we can see the total amount of traffic passed the core link, however, we cannot distinguish how much portion of the consumed bandwidths are allocated to each edge-to-edge traffic flows.

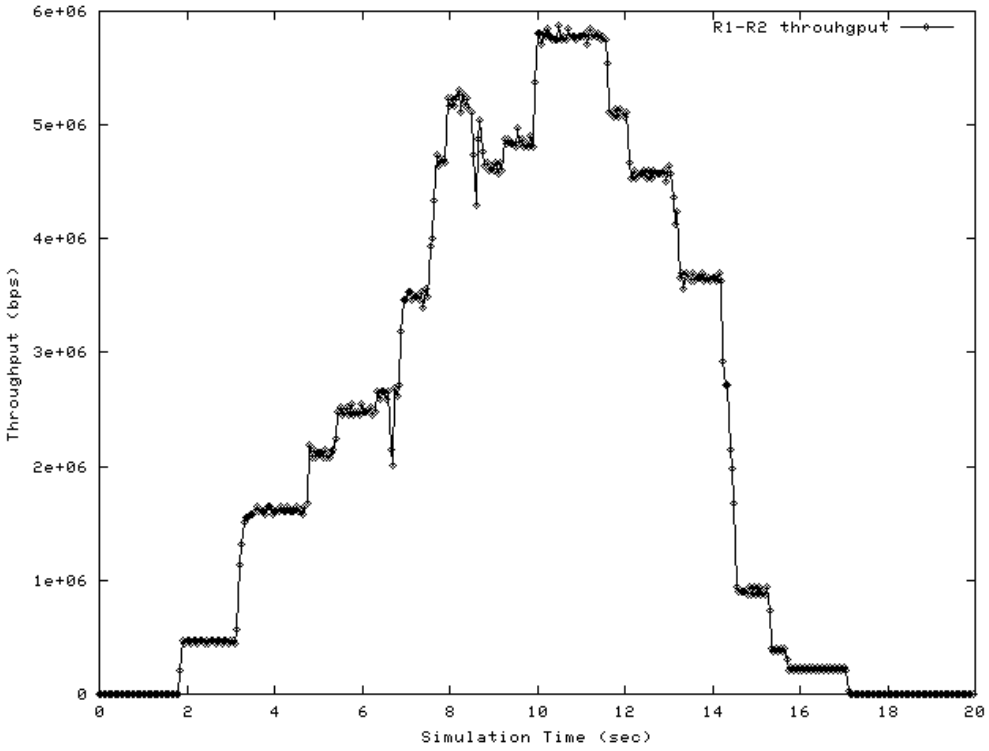


Figure 24. Aggregated Throughput at R1-R2 Link

With the proposed edge-to-edge QoS monitoring methods, the amount of bandwidth used for each edge-to-edge DiffServ flow can be bounded by min/max values (from the first method) and further can be exactly counted (from the second method). The total amount of traffic is classified and distinguished according to four different edge-to-edge DiffServ flows. In Figure 25, we can see that the actual amount of edge-to-edge DiffServ is located between the min/max bound. Network administrators can understand how different edge-to-edge DiffServ flows are aggregated to the total amount of traffic at each network link from the QoS monitoring methods.

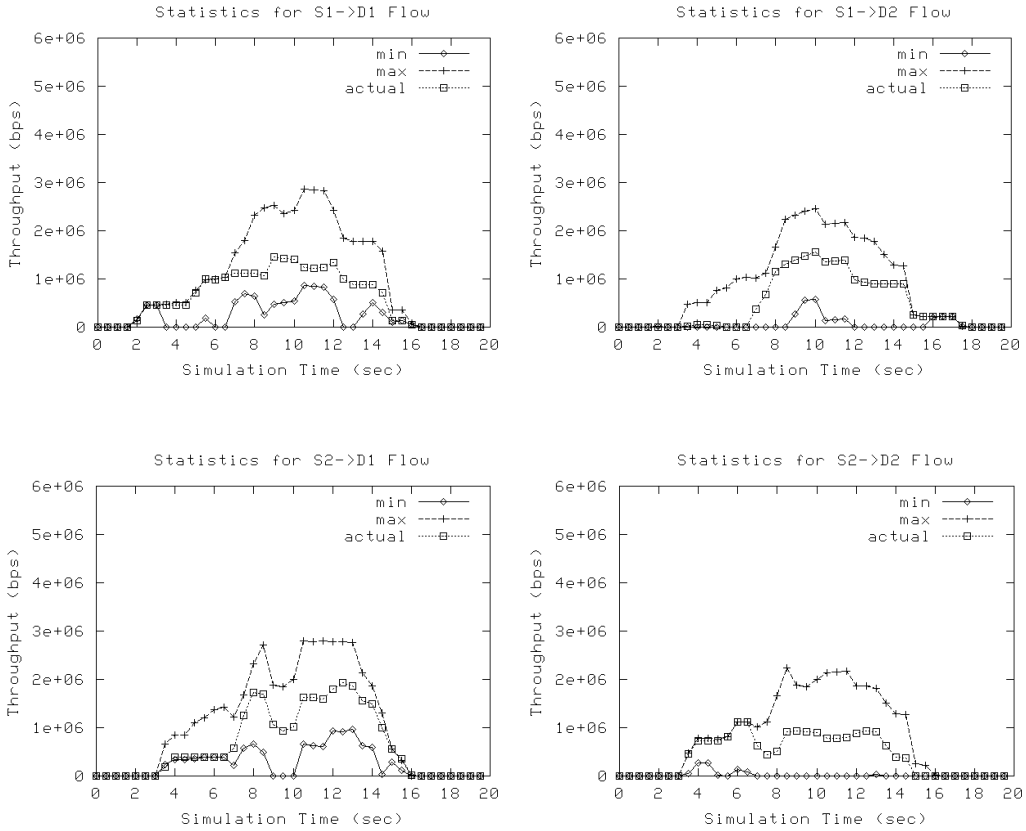


Figure 25. Statistics for Four Different Edge-to-Edge DiffServ Flows

5.2 Dynamic Adaptive Provisioning

As previously stated, DiffServ lacks in providing concrete provisioning methods to configure QoS parameters in DiffServ devices. The simplest approach is static provisioning method that reserve QoS parameters once and do not change the value during considerable amount of time period. Although the static provisioning approach is easy to implement, generally the static provisioning method severely underutilize the overall bandwidth and also inflexible to dynamic network changes.

Dynamic adaptive provisioning is an alternative approach to configure QoS parameters in each network device periodically to reflect dynamic network status changes. In order to enable dynamic adaptive provisioning, it is required to monitor current network status repeatedly. With the proposed edge-to-edge QoS monitoring methods, the dynamic adaptive provisioning on DiffServ domains can be realized with easy and efficient mechanisms.

For example, when an expedited forwarding service is provisioned, the static provisioning method reserves bandwidth for all edge-to-edge paths. In the worst case, this provisioning needs that every link reserves the amount of EF bandwidth times number of edge routers because a certain link might provide network bandwidth for every edge-to-edge DiffServ flows. However, if we consider the routing paths of edge-to-edge DiffServ flows and actual traffic demand monitored by the proposed monitoring methods, the required bandwidth can be decreased considerably. While the worst case static provisioning underutilizes most of link bandwidths, the dynamic adaptive provisioning with the monitoring information can efficiently utilize the valuable bandwidths.

For the assured forwarding service, the edge-to-edge QoS information can be used effectively as well. When provisioning assured forwarding service, we need to have edge-to-edge QoS because the network service is established on the edge-to-edge basis. If an AF service experiences packet drops at some network links, edge-to-edge QoS information can report how much total packet drops for each edge-to-edge DiffServ flow, and with this information, network administrators can make a traffic engineering decision that changes traffic routes or provisions the additional bandwidth to appropriate links.

5.3 Bottleneck Detection and Resolution

Insufficient provisioning and instantaneous traffic burst results in bottleneck problems at some network links. When a bottleneck happens, every traffic flow

passing through the overloaded network link suffers from unexpected packet drop, delay, and jitter, which cause QoS degradation of network services. In order to provide guaranteed QoS to network customers, network administrators want to detect bottleneck points and resolve possible reasons of the bottleneck. Edge-to-edge DiffServ flow monitoring can help network administrators for this purpose.

Network administrators can easily detect bottleneck points when bandwidth consumption of a network link reaches to the maximum of reserved bandwidth and packet drops at the link increases considerably. However, it is hard to decide what kinds of operations should be performed to alleviate the bottleneck problems because current DiffServ framework does not distinguish traffic flows in a DiffServ class. With the edge-to-edge DiffServ QoS monitoring information, the network administrators can find how many different edge-to-edge DiffServ flows exist in current overloaded traffic aggregates and how much portion of the reserved bandwidth are consumed by each edge-to-edge DiffServ flow. When every edge-to-edge DiffServ flow competing for the reserved bandwidth is identified, there can be various resolution steps depending on provisioning and service policies within the DiffServ domain. For example, an edge-to-edge DiffServ flow consuming the greatest amount of reserved bandwidth is identified and the ingress and egress edge routers of the edge-to-edge DiffServ flow and routing path between the two edge routers are easily found. If the resolution policy is to decrease the amount of bandwidth usage of the greatest edge-to-edge DiffServ flow, proper shaping operation is performed at the identified ingress router and decreases the amount of incoming traffic of the edge-to-edge DiffServ flow.

In order to show how the proposed monitoring methods detect the bottleneck link and resolve the problem, another simulation has been performed with the same network configuration as in Figure 23, but with different traffic pattern. One of four edge-to-edge DiffServ flows is assumed to consume most of link bandwidth in this simulation. However, since different edge-to-edge DiffServ flows are aggregated into the same DiffServ class in core routers, it is not

distinguishable which flows are causing the saturation at the bottleneck link of R1→R2. Figure 26 shows the aggregated throughput measured at the bottleneck link within the simulation period.

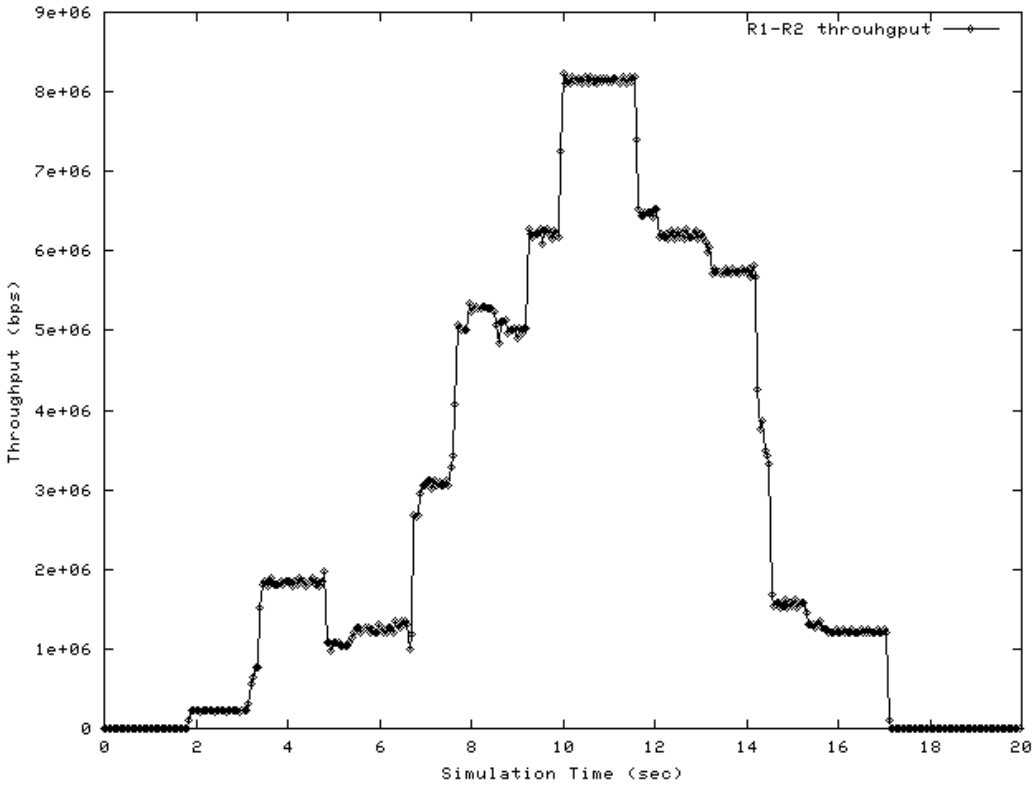


Figure 26. Aggregated Throughput at a Bottleneck Link

With the proposed monitoring methods, the aggregated throughput can be analyzed and divided into four different edge-to-edge DiffServ flows as in Figure 27. Among four edge-to-edge DiffServ flows, we can detect that the edge-to-edge DiffServ flow from S1 to D2 is consuming the most of bandwidth at the bottleneck link. When the most dominant edge-to-edge DiffServ flow is specified, various resolution actions might be executed. Admission control on the edge-to-edge DiffServ flow at the ingress router is one immediate remedy to alleviate the saturation. Rerouting is another option if there is detour routing path without

crossing the bottleneck link and dynamic adaptive routing policy is available. If the bottleneck problem is occurring constantly in long-term monitoring results, bandwidth re-provisioning or SLA re-negotiation might be more effective solutions.

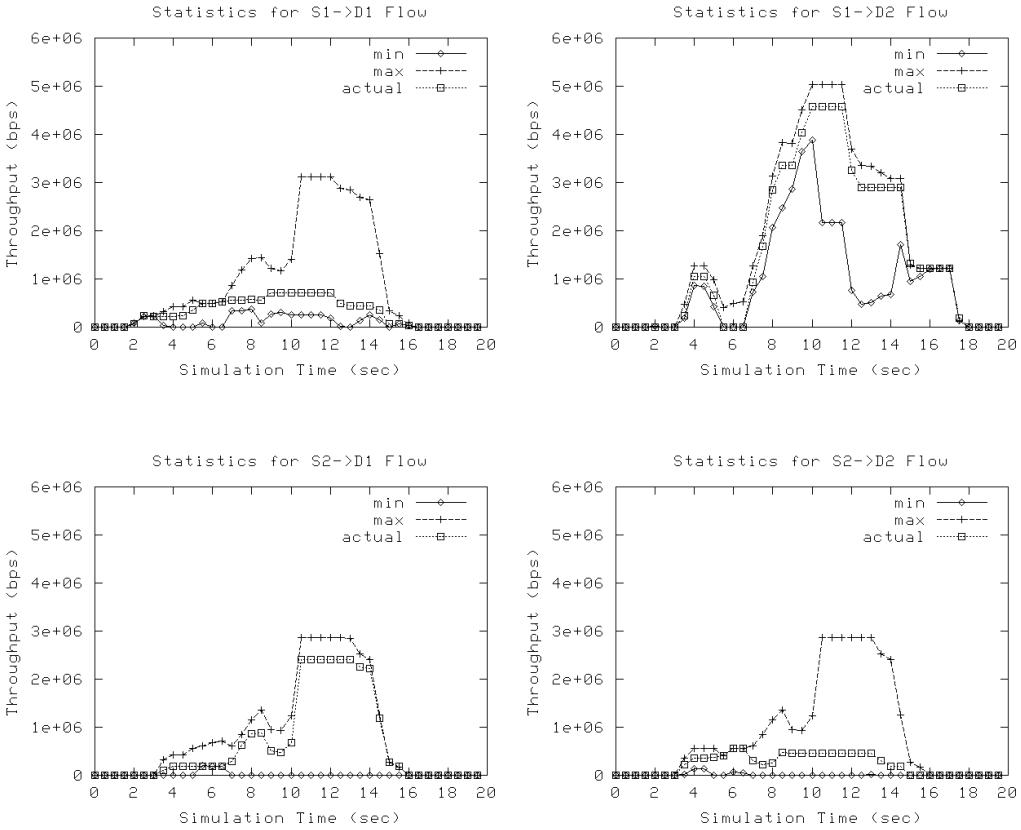


Figure 27. Edge-to-Edge Monitoring Detects S1 → D2 Dominant Traffic

5.4 Other Applications

The edge-to-edge QoS information monitored by the proposed monitoring methods can be applied to many other management functions in DiffServ domains. In this section, several other applicable issues are discussed.

In Section 3, PDB has been explained as a term to describe edge-to-edge behavior of a traffic flow within a DiffServ domain. The PDB is used to notate required QoS parameters for a whole domain, not for each DiffServ router in the domain. Since PDB is only a specification of a service across a DiffServ domain, it is required to measure real QoS status over a whole DiffServ domain. The edge-to-edge DiffServ flow information measured by the proposed monitoring method can be a good indication whether the current service status meets the specified PDB or not.

Edge-to-edge QoS information can be used for intradomain management of DiffServ domains. When an end-to-end connection is established across multiple different DiffServ domains, a kind of management operations to coordinate service quality among different DiffServ domains. Bandwidth broker architecture is one of the proposed management architectures in intradomain DiffServ management. In order to negotiate the service status of each DiffServ domain, the bandwidth broker needs to have current network status of its own DiffServ domain. Edge-to-edge QoS information is valuable information for this purpose. Since edge-to-edge QoS information provides detailed QoS status from one edge router to another edge router, it is helpful for bandwidth brokers to negotiate QoS parameters for a certain end-to-end connection. For a certain end-to-end connection, the ingress edge and egress edge of a certain DiffServ domain can be identified and the current connection status between the two edge routers is easily reported by the proposed monitoring methods.

In addition, edge-to-edge DiffServ QoS information can be used as a building block for an Internet pricing system. It is essential to have accounting information to impose usage fees on the service received. Original DiffServ framework does not distinguish edge-to-edge DiffServ flows and it is impossible to impose usage fees on the amount of bandwidth consumed. Only flat-rate pricing model is possible. However, with the proposed edge-to-edge QoS monitoring information, it is possible to distinguish aggregated flows with different ingress / egress pairs and to impose different service fees on the amount of bandwidth used in a

DiffServ class. If the edge routers can distinguish different user groups at access networks, more fine-grained pricing model can be developed with combining the additional user information and QoS monitoring information. Moreover, SLA negotiation with customers is also benefited by the edge-to-edge QoS information since the information contains actual bandwidth consumption between two edge routers.

6. Edge-to-Edge DiffServ Flow Management System

An SNMP-based edge-to-edge DiffServ flow management system based on the proposed monitoring methods has been developed. The network status with the edge-to-edge status information can be provided to network administrator for easy understanding of traffic snapshot of a DiffServ domain. The provided information can be used for traffic engineering decisions.

6.1 SNMP-based Network Management

Simple Network Management Protocol (SNMP) framework is chosen for the DiffServ management platform. SNMP is simple, cost-effective, and the *de facto* standard for managing Internet-related systems. SNMP is a standard protocol suite for the Internet network management. Internet Engineering Task Force (IETF) first standardized the protocol and initiated SNMP-based management. The original targets for this effort were TCP/IP routers and hosts. However, the management architecture is inherently generic so that it can be used to manage various types of systems [70, 71, 72, 73, 74, 75, 76, 77].

Since SNMP had the philosophy of simplicity and efficiency, it has been widely accepted and deployed in many areas of network and service management. Management targets have been extended from the simple network devices to the complicated network servers. Almost all internetworking vendors have developed and are marketing SNMP products.

The SNMP network management model consists of the following elements [70, 79]. Figure 28 illustrates conceptual relations between the elements. SNMP management defines two network entities, managing system and managed system, and the communication methods between two entities.

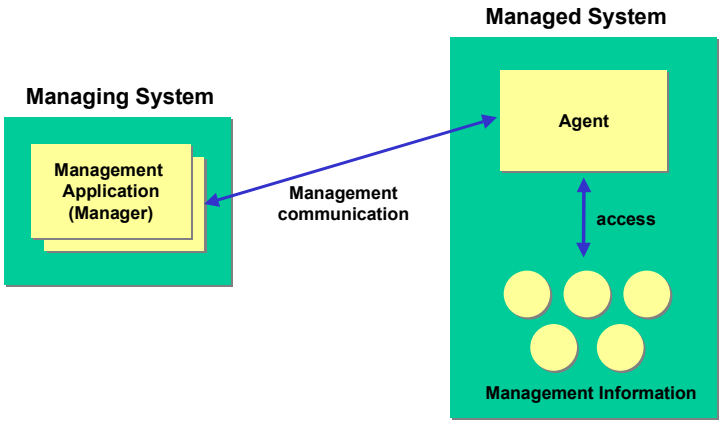


Figure 28. SNMP Management Model

Managing system contains one or more processing elements called management applications, shortly, managers. Manager performs management functions over managed nodes it controls. Each managed system has a processing entity called an agent which gathers various management information from the managed system. SNMP itself is a set of communication methods between manager and agent. The protocol defines three basic operations, GET, SET, and TRAP. The GET operation is initiated by a manager to retrieve management information from an agent. When the manager needs to change management information in managed systems, the manager performs the SET operation on the specified agents. The TRAP operation is an unsolicited communication from agents to managers. Agents send TRAP information to specified managers when managed system initiates any events. The SNMP operations between manager and agent are depicted in Figure 29 [78].

Management information and events used in the SNMP management model should be clearly defined in predetermined formats. The Structure of Management Information (SMI) [71] includes the model of management information and events, the allowed data types, and the rules for specifying management information and events. It sets the rules for how management information is

described and stored. Management Information Base (MIB) [76] is a set of related management information, events, and implementation compliance requirements following SMI rules. A MIB represents a collection of managed objects and each managed object can be managed remotely by managing MIB information via SNMP operations. MIB information is written in the subset of Abstract Syntax Notation ONE (ASN.1) [78]. When a networked system is to be managed by SNMP, the first thing to do is to define MIB for the system.

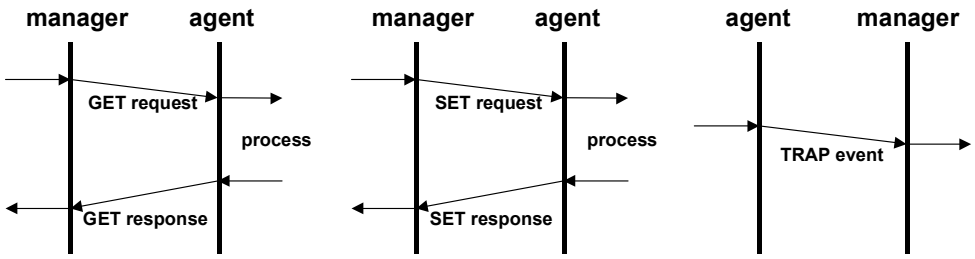


Figure 29. SNMP Manager-Agent Interactions

6.2 SNMP-Based DiffServ Flow Monitoring

Management of a DiffServ network starts from managing a DiffServ router in its domain. The Simple Network Management Protocol (SNMP) framework is chosen for the DiffServ management platform. SNMP is simple, cost-effective, and *de facto* standard for managing Internet-related systems. In this section, it is explained how to manage a DiffServ router by using DiffServ MIB from the IETF.

The IETF DiffServ working group currently suggests an SNMP Management Information Base (MIB) for DiffServ architecture [17]. The MIB is designed according to the DiffServ implementation conceptual model [14] for managing DiffServ routers in the SNMP framework. The initial draft was proposed in July 1999 and several extensions were made. Detailed definitions are still being elaborated and extended in the working group. Table 4 summarizes the primary object tables defined in the latest DiffServ MIB.

Table 4. DiffServ MIB Structure

Element	Table Name	Description
Classifier	Classifier	general classification parameters
	SixTupleClfr	5-tuple information + DSCP value
Meter	Meter	general metering parameters
	TBMeter	token bucket meter
Action	Action	general action parameters
	MarkAct	marker action
	CountAct	counter action
	AbsoluteDrop	absolute drop action
Queue	AlgDrop	algorithmic dropper parameters
	RandomDrop	random dropper parameters
	Queue	queuing parameters
	Scheduler	scheduling parameters

The DiffServ MIB tables are categorized in four architectural DiffServ elements, which are classifier, meter, action, and queue. Each logical element contains several MIB tables and specific MIB objects for describing TCBs in a DiffServ router. The shaded MIB tables in Table 4 are subordinate tables that belong to the general element table. A ‘specific’ table entry in a general element table points to an entry in the subordinate table. This mechanism enables the DiffServ MIB more flexible and extensible to adopt additional functions. The general element table contains only the general parameters while the subordinate table contains specific, detailed, and supplementary parameters. A Classifier element has a list of classifiers a router handles. Six-tuple information (5-tuple information and a DSCP value) is recorded in the SixTupleClfr table and referenced by the Classifier table. The Meter table redirects packets according to the metering result. Currently, only the token bucket meter is available as a specific meter. An entry in the Action table can select one of three actions defined

in the Action element. They are marker, counter, and absolute dropper. A Queue element is used for describing queue-related operations in DiffServ TCBs. The Algorithmic dropper, Queue, and Scheduler are three different tables that perform different queue operations.

The DiffServ table entries are linked to each other with the RowPointer textual convention. A RowPointer object is used for pointing an entry in the same or a different table [16]. The DiffServ MIB represents a TCB as a series of table entries linked together by RowPointers. With this scheme, many different TCBs can be efficiently represented in the six tables. For example, several TCBs have the same classifiers, the same actions or meters on the same queues. Thus, the same parameters need not be defined separately. The current MIB design provides parameter sharing for different TCBs. If the DiffServ MIB defines a TCB table that contains entries for all TCBs, the size of the table would be much more than the sum of the current table entries. Figure 30 shows the RowPointer linked relationship of tables in the DiffServ MIB.

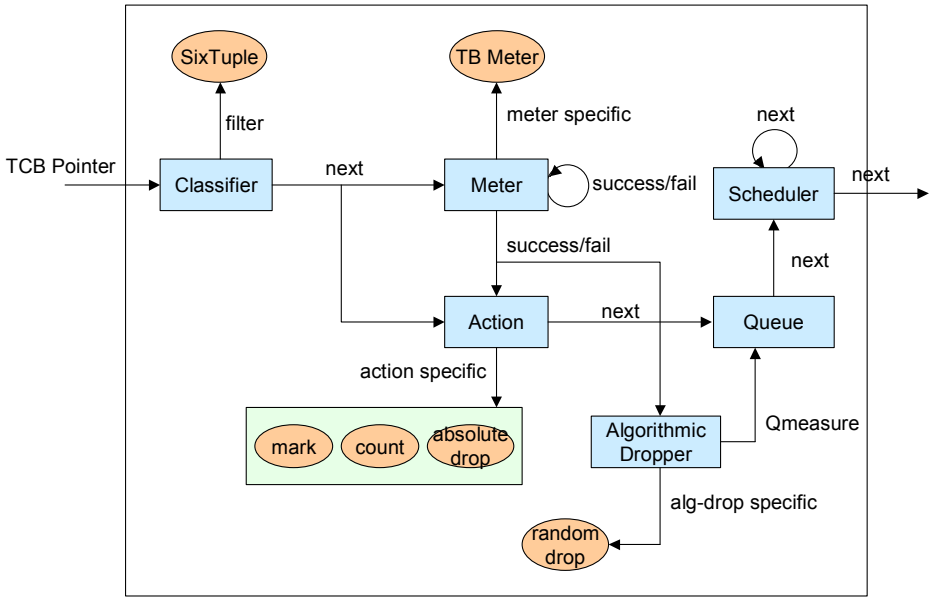


Figure 30. DiffServ MIB Table RowPointer Linked Relationship in a TCB

The rectangle boxes represent major DiffServ element tables explained previously and oval shapes represent subordinate tables belonging to one of the major element tables. From the Classifier table to the Scheduler table, each table entry is linked to compose a TCB. By following the link path, we can comprehend how the classified packets are handled and treated in a TCB.

In order to make the DiffServ MIB available to management systems, an SNMP agent that fills the values of the DiffServ MIB should be incorporated into a DiffServ router. As in Figure 5, an SNMP agent resides as a configuration and monitoring module in a DiffServ router, which controls and retrieves the DiffServ parameters defined in the DiffServ MIB. The routing module can also be managed by the SNMP agent, with MIB II or appropriate MIBs supporting routing protocol parameters. Since the implementation details of packet handlings vary with the system architecture, different methods are required to obtain and set DiffServ parameters among different DiffServ router implementations. However, the management modules should not affect the internal packet handling functions because management functionality is not the fundamental role of routers.

However, the current DiffServ MIB exists only for managing the characteristics of a single DiffServ router. It does not provide a complete network picture of a set of DiffServ routers in one administrative domain. In order to provide such high-level management functions, the single-router management framework should be extended.

6.3 Construction of Edge-to-Edge DiffServ Flows

An edge-to-edge DiffServ flow is required to have topology and performance information. Topology information is constructed from the routing tables in each router and performance information is constructed from DiffServ MIB [17] values in each DiffServ router. Constructing edge-to-edge DiffServ flows thus consists of two phases, as shown in Figure 31. First, the topology generator produces

topology information as a linked list of a set of routers and the performance analyzer aggregates performance parameters of each router in the routing path by using topology information. MIB II [18] and DiffServ MIB are used to construct the edge-to-edge DiffServ flows.

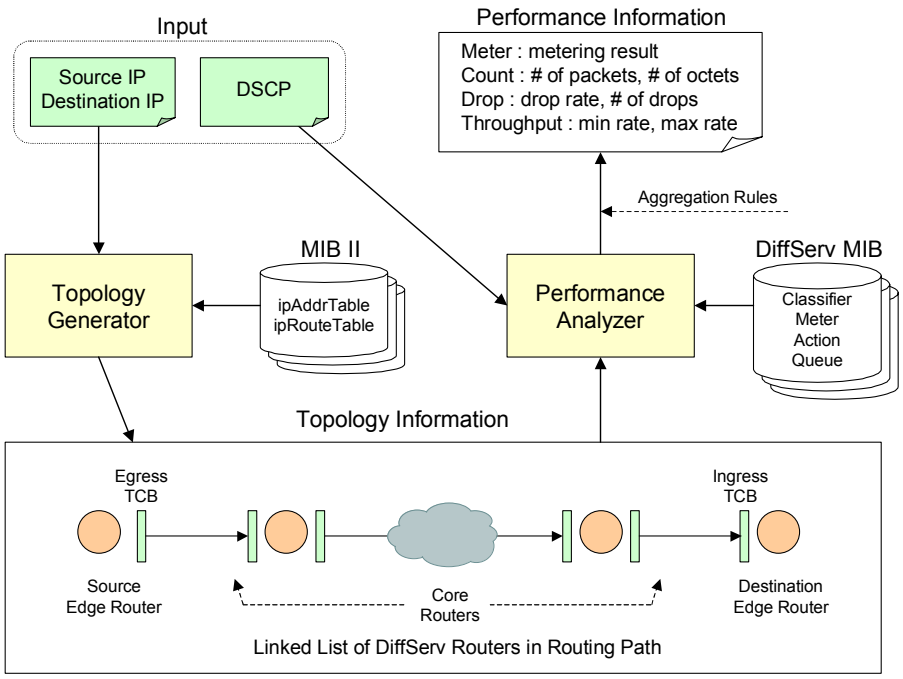


Figure 31. Construction of Edge-to-Edge DiffServ Flows

Since each DiffServ router supports routing protocols, the router keeps a routing table that contains a list of the next hop routers for a given destination IP address. The MIB II standard has MIB objects for containing the routing table. A central SNMP manager can retrieve the routing table information to construct a whole routing connectivity map in a DiffServ domain. Two MIB tables, an ipAddrTable and an ipRouteTable, can be used to create topology information. The ipAddrTable contains IP addresses of all network interfaces in a router and the ipRouteTable contains the IP routing table with the next hop host and network interface for a set of destination IP addresses. By combining the two table entries

we can obtain every source-to-destination routing path. Given the source and destination IP addresses, the topology generator outputs a linked list of DiffServ routers composing the routing path.

Performance information of edge-to-edge DiffServ flows is obtained from the DiffServ MIB. Each DiffServ router has performance parameters observed locally. The parameters include metering parameters, counter values, numbers of dropped packets, minimum and maximum rates of packet transmission, and so on. These parameters are calculated and maintained for each DSCP value; that is, the DiffServ MIB of a DiffServ router contains all the performance parameters of DiffServ flows it processes. When a linked list of routers composing a DiffServ routing path is given, the performance analyzer aggregates the values of the parameters from each DiffServ router one by one and produces edge-to-edge performance information of a DiffServ flow.

6.4 Design of an Edge-to-Edge DiffServ Flow Management System

The architecture consists of three distinct layers, as depicted in Figure 32. The three-tier architecture includes a network management system (NMS) client running in a Web browser, an NMS server containing a Web server and DiffServ manager, and network elements performing DiffServ routing and SNMP management.

The NMS server is a central server for managing a set of DiffServ routers and providing management interfaces to a set of Web browsers. The Web server located in the NMS server layer provides a Web-based management interface in Web browsers. The integration of the Web server and the DiffServ manager can be accomplished in various ways such as a basic HTML file access method, a Common Gateway Interface (CGI) method, and a Java applet/servlet method.

The DiffServ manager performs three high-level DiffServ management functions, which are configuration management, metering and monitoring, and

end-to-end flow management. The management database is used for storing and retrieving the combined and analyzed data from the MIB II and DiffServ MIB. At the bottom of the DiffServ manager, an SNMP manager communicates with a set of SNMP agents running in different DiffServ routers within a DS domain.

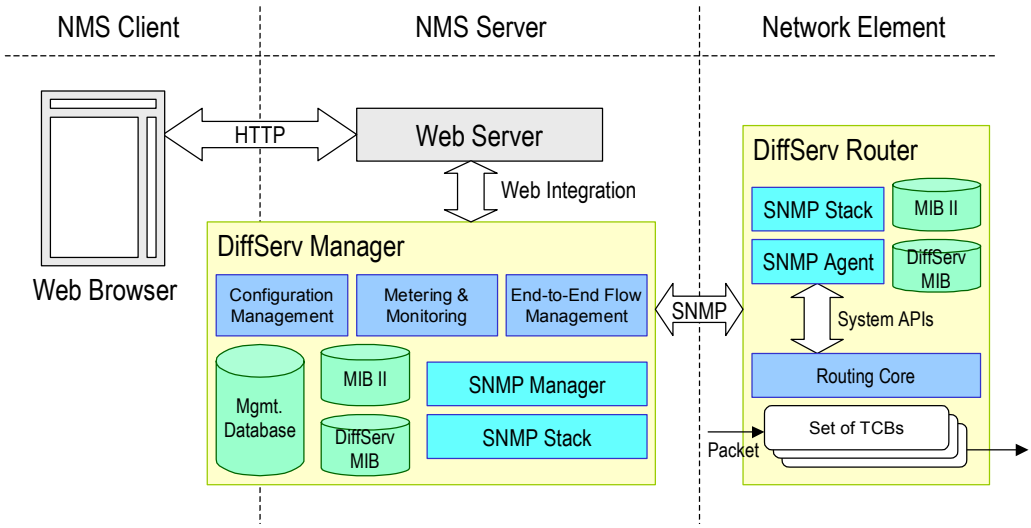


Figure 32. Design Architecture of the DiffServ Management System

Three high-level DiffServ management functions perform sophisticated and extended management functions. Configuration management function performs remote configuration provisioning. Every DiffServ parameter is determined and enforced via the configuration management function. Metering and monitoring function periodically observes the status of DiffServ routers and compares the results with predefined desirable performance metrics. Such conformance test results are necessary for modifying behaviors of a DiffServ router. Flow management function summarizes all the DiffServ flows in a DS domain and provides the end-to-end DiffServ flow characteristics. The function collects routing tables and DiffServ flow information and constructs overall end-to-end parameters of each DiffServ flow.

DiffServ routers are managed network elements in the design architecture. A DiffServ router contains a routing core module to control a set of TCBs that

execute packet forwarding according to various DSCP values, and an SNMP agent module to handle SNMP manager requests for the DiffServ MIB. System-dependent APIs are used to connect the SNMP agent module and the routing core module. The values of DiffServ MIB variables are determined by specific system-dependent system calls. The methods of retrieving and setting DiffServ parameters in the routing core module need not be the same among different implementation architectures.

Within a DiffServ domain, numerous DiffServ routers and DiffServ management clients interwork with each other. The three-tier architecture offers distinct advantages in such environments. One centralized DiffServ manager controls a set of DiffServ routers while providing management interfaces to a set of management clients at the same time. However, by separating the management user interfaces from the manager itself, the DiffServ manager is able to concentrate on management functions and thus the performance of the DiffServ manager can be improved.

6.5 Linux-Based Edge-to-Edge DiffServ Flow Management System

Linux, a shareware operating system, supports QoS features in its networking kernel from the kernel version 2.1.90 [21]. The QoS support offers a wide variety of traffic control functions, which can be combined in a modular way. Based on this Linux traffic control framework, W. Almesberger et al. have designed and implemented basic DiffServ-field classification and manipulation functions required by DiffServ network nodes [22]. The extended DiffServ features are freely available in the form of a kernel patch. The latest DiffServ package (DS-8) [23] contains the kernel patch, a control program (tc), and several PHB scripts written with the control program. By installing the DiffServ package, a Linux system is able to perform DiffServ router functions.

A testbed has been established with a set of DiffServ routers in the Linux

systems and the systems have been added an SNMP agent for the DiffServ MIB in each router. A centralized DiffServ domain manager which includes an SNMP manager for the DiffServ MIB and MIB II has been implemented in a dedicated Linux system as well. A Java-based DiffServ management console has also been developed for delivering various management services to human administrators. Figure 33 shows the implementation architecture of our DiffServ domain manager.

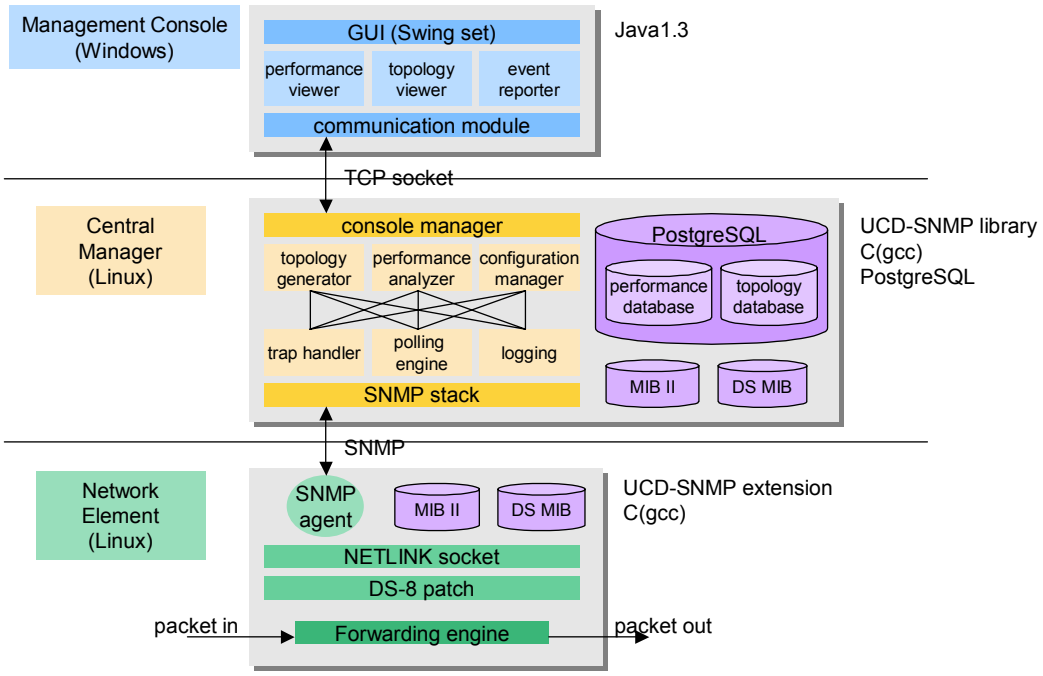


Figure 33. Implementation Architecture of DiffServ Domain Management System

A DiffServ domain manager includes both the management console and the central manager together. The management console is responsible for supporting management operations to human administrators, while the central manager executes real management operations. By separating management roles into two modules, each module can concentrate on its own functionality.

The central manager is implemented in the Linux platform with C language. UCD-SNMP [24] management APIs are used for handling SNMP manager

operations. The central manager can configure, monitor, and report the characteristics of DiffServ routers and DiffServ networks. A set of DiffServ edge-to-edge traffic aggregates information is constructed by following the method in Section 4 and stored in a PostgreSQL database of version 7.0.2 [25]. The central manager sends SET and GET requests to DiffServ agents in DiffServ routers under its control in order to record current MIB II and DiffServ MIB values. The PostgresSQL database is used for storing performance and topology information derived from MIB tables. The database also contains configuration data for supporting management consoles. Topology generator, performance analyzer, and configuration manager are three distinct functional modules implemented in the central manager.

The management console is developed in Microsoft Windows platform in our implementation. However, since the implementation language is Java, the runtime module is not dependent on specific platforms. If Web integration is highly recommended for ubiquitous access, the management console can be easily adjusted to Java applets running in Web browsers. However, standalone Java applications show a quicker and more stable execution performance than Java applets running in Web browsers.

Communications between the management console and the central manager follows proprietary application protocols that have been developed over TCP sockets. Management operations and response data are transferred by the protocol. The protocol also supports multiple concurrent management consoles and password security for providing different management views to different human administrators.

A DiffServ agent is an SNMP agent with MIB II and DiffServ MIB running on the Linux DiffServ router. Basically, the agent extracts DiffServ parameters from the Linux traffic control kernel and modifies the appropriate MIB values on a request from the DiffServ manager. The agent also receives management operations from the DiffServ manager and performs the appropriate parameter changes in the Linux traffic control kernel.

The organization of our Linux DiffServ router implementation is explained in Figure 34. There are two process spaces in the Linux operating system, the user space and the kernel space. Extending from Linux traffic control framework, the Linux DiffServ implementation resides in the kernel space. In the user space, the DiffServ SNMP agent is implemented. Communication between the DiffServ agent and the Linux traffic control kernel is effected via NetLink sockets [20]. The NetLink socket is a socket-type bidirectional communication link located between kernel space and user space. It transfers information between them.

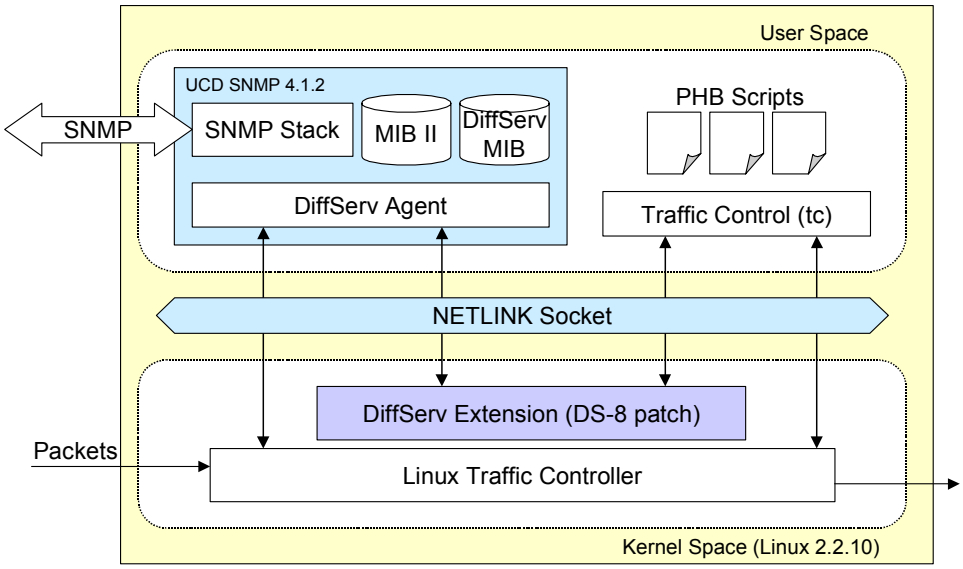


Figure 34. Organization of Linux DiffServ Router Implementation

The organization of our Linux DiffServ router implementation is as follows. There are two process spaces in the Linux operating system: the user space and the kernel space. Extending from the Linux traffic control framework, the Linux DiffServ implementation resides in the kernel space. In the user space, the DiffServ SNMP agent is implemented in combination with the Linux traffic control program. Communication between the DiffServ agent and the Linux traffic control kernel is effected via NetLink sockets [26]. The NetLink socket is a

socket-type bidirectional communication link located between kernel space and user space. It transfers information between them.

The agent has been implemented by using UCD SNMP agent extension package. UCD SNMP 4.1.2 provides the agent development environment. The DiffServ agent uses the traffic control program (tc) or NetLink socket directly for accessing DiffServ parameters in kernel space and manipulates the values of MIB II and DiffServ MIB.

When a testbed with multiple DiffServ routers is organized, each DiffServ router should include SNMP-based QoS agent with proposed MIBs. The DiffServ domain manager retrieves local QoS information monitored in each DiffServ router via SNMP polling and trap operations. Human administrators look up the analyzed monitoring information from console viewer remotely and effectively. The overall system architecture on a testbed network is depicted in Figure 35.

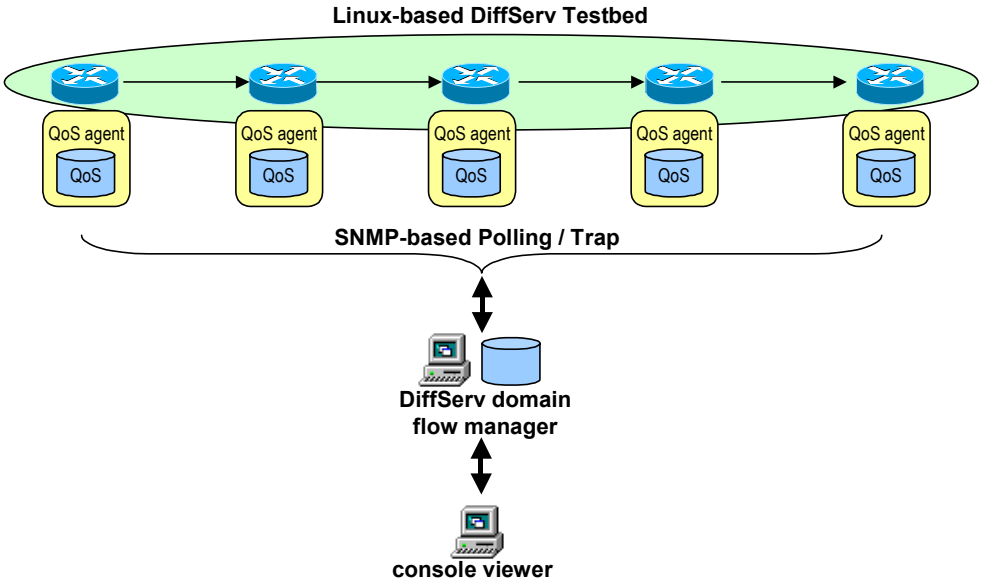


Figure 35. Operational Architecture on a Testbed Network

The centralized DiffServ domain manager can be distributed with multiple

instances of the domain manager if the amount of network-wide statistics is too huge to be processed in one central manager. For faster and more efficient analysis of the statistical data, it is needed to optimize computations performed in the DiffServ domain manager. However, these considerations can be different among different implementations and not within the scope of this thesis.

Some screenshots from the console viewer working on a testbed DiffServ network are showed in Figure 36 to Figure 39. The testbed is composed of six Linux-based DiffServ routers and five direct network connections among them as in Figure 36. Four DiffServ routers (lena, anne, charles, and jordan) are edge routers and two DiffServ routers (don and teri) are core routers. All six routers organize a DiffServ domain. Test traffics are generated from the outside of four edge routers according to predefined traffic generation scenarios.

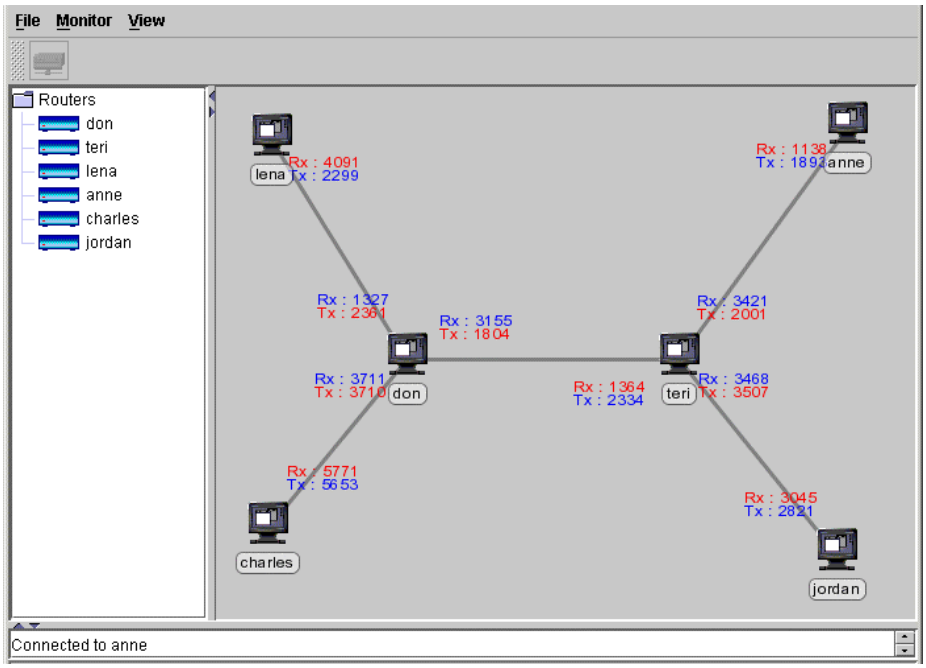


Figure 36. Main Monitoring Console on a Testbed Network

Figure 36 shows main monitoring console network administrators can see. On the left window, all DiffServ routers under control are listed, and the connectivity

among the routers is graphically represented on the right window. The numbers written at each router icon mean the local throughput measurements values. Each DiffServ router keeps track of the numbers of incoming and outgoing traffic at each network interface it has. The icons on the right window are moveable so that network administrators can locate each router icon where they want and change the appearance of topology.

When network administrators want to see network statistics of a certain DiffServ flow of a QoS class, screenshots such as Figure 37 will appear. At each monitoring time period, the central manager records network statistics of each DiffServ class from each DiffServ router. In Figure 37, you can see there exist two traffic sources (lena and charles) and two traffic sinks (anne and jordan) at this monitoring time period.

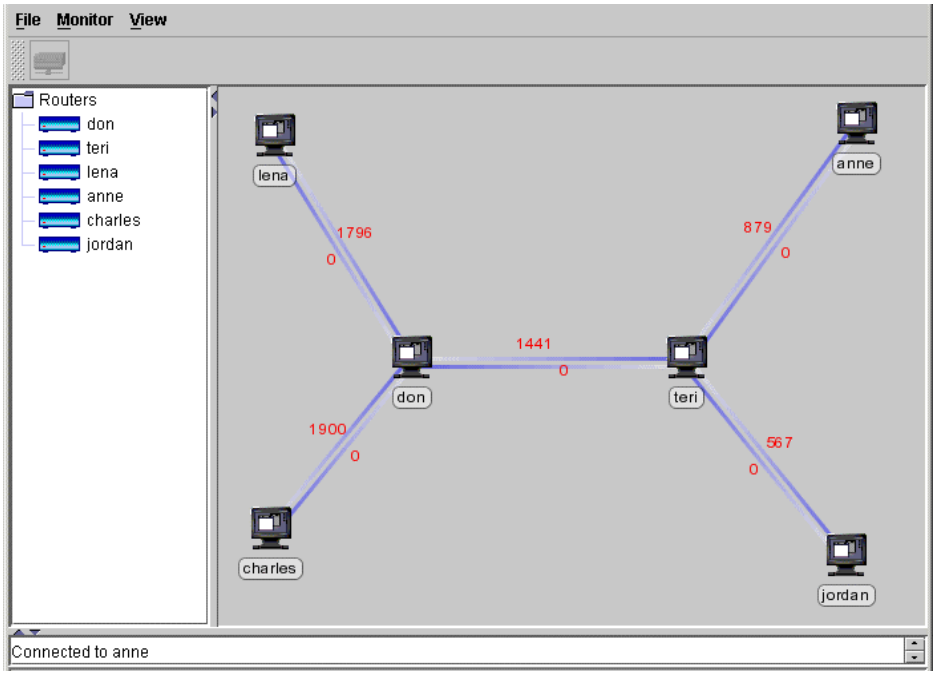


Figure 37. Total DiffServ Flows of a Class at a Certain Time

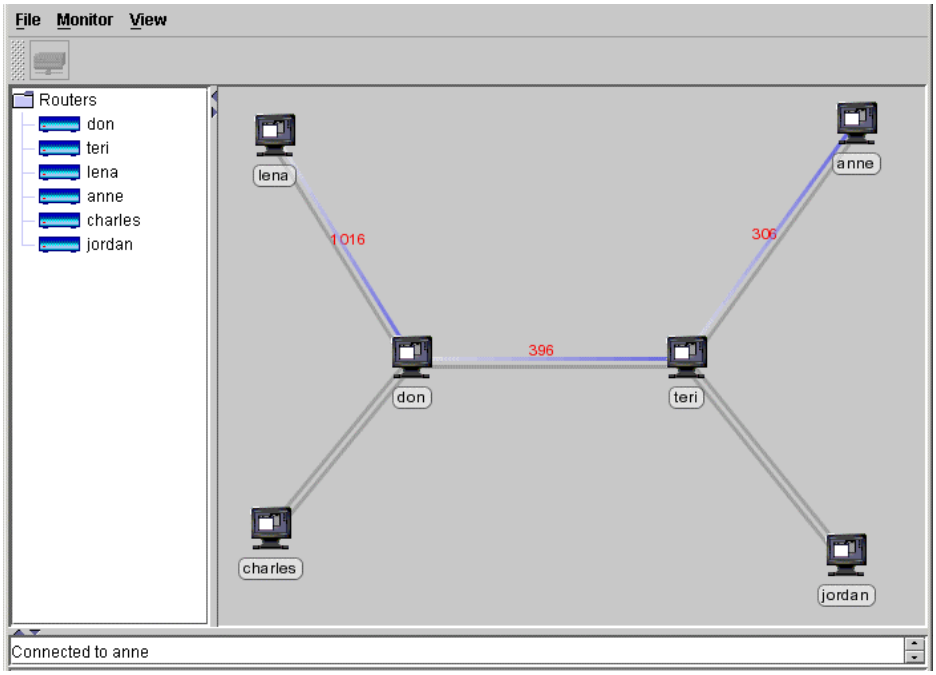


Figure 38. Edge-to-Edge DiffServ Flow from lena to anne

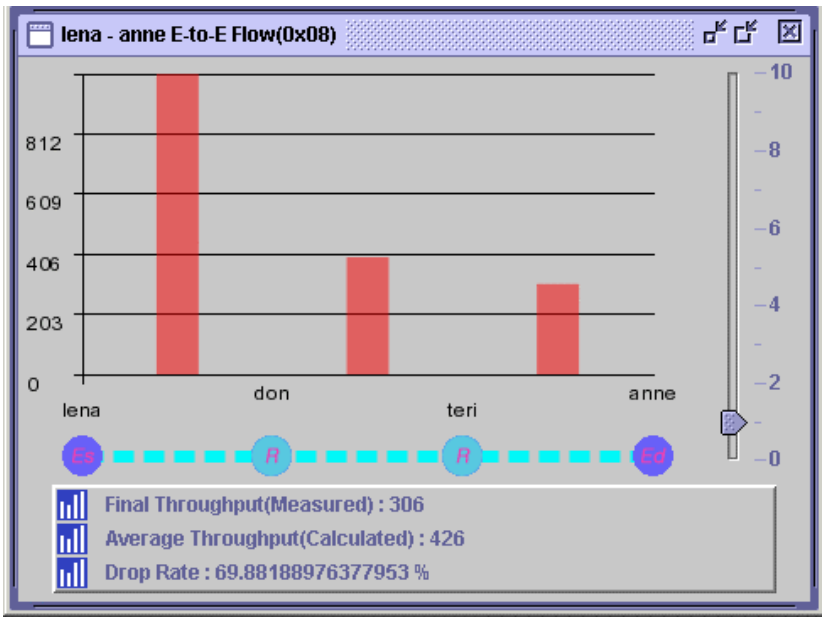


Figure 39. Edge-to-Edge DiffServ Flow Statistics

When network administrators selects one source edge router and one sink edge router from Figure 37, the edge-to-edge DiffServ flow statistics are calculated by following the QoS concatenation rules described in Chapter 4 and the results are showed as in Figure 38. The statistics of the edge-to-edge DiffServ flow from the system ‘lena’ to the system ‘anne’ are calculated and represented in the topology window. The statistics are also represented in a histogram as in Figure 39 to keep record of dynamic statistical changes. Network administrators can understand the temporal behavior of the edge-to-edge DiffServ flow and pinpoint bottleneck points efficiently from the monitoring window.

7. Related Work

When realizing the Internet QoS frameworks, network providers need to have a well-defined management methods and architectures. In this chapter current research work related to IP QoS monitoring and management approaches in detail.

On the subject of managing QoS-enabled IP networks, there has been only few research work yet [25]. Current IP QoS framework is mainly focusing the provisioning and configuration mechanisms, and leaving much work to be done for complete QoS management. Network QoS monitoring has an important role when building complete network QoS management systems.

7.1 Monitoring Real-Time Network Traffic

In order to manage QoS of network flows, measuring and monitoring status of current network traffic is the first thing to do. There are many tools and research work to monitor real-time network traffic from a network interface of monitoring system. Various commercial systems and open-source systems provide the capability of monitoring real-time network traffic. Some are software-based and some are hardware-based. Advanced network devices tend to have such monitoring capability as a built-in module.

The well-known implementation approach for this purpose is using system-independent libpcap programming interface developed by Network Research Group at the Lawrence Berkeley Laboratory [80]. libpcap is a packet capture library containing a set of uniform library interfaces to the OS-dependent packet capture systems. Since libpcap standardizes packet capture features previously not interoperable among different systems, various real-time packet monitoring systems have been easily developed. There are numbers of systems using libpcap library, such as tcpdump [81], ethereal [82], and ntop [83]. However, the packet

capturing utility, in general, only shows the contents of network packets and does not provide well-defined management functionality.

As the SNMP-based network management has been widely accepted in IP-based network environment, the simple and efficient management framework started being added to real-time monitoring systems. SNMP-based network monitoring architecture separates packet capturing and packet analysis as in Figure 40. The real-time packet capturing module extracts packet header information and stores it in the format of predefined MIB. The SNMP agent reports the statistical data stored in the MIB to the SNMP manager when the manager requests them. With the SNMP framework, remote network status monitoring with more advanced management functions is enabled.

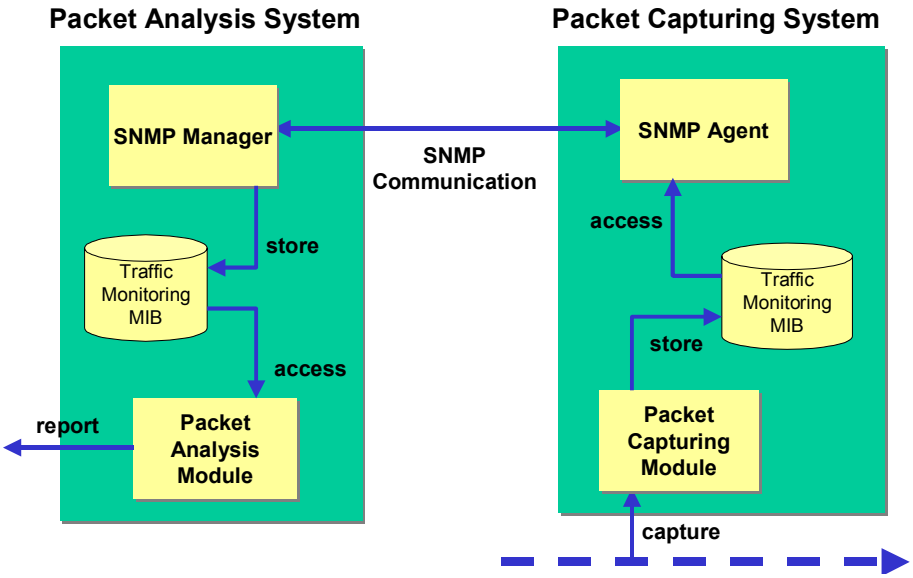


Figure 40. SNMP-based Traffic Flow Management Architecture

IETF's real-time flow measurement (RTFM) architecture [6, 7] and suggests and implements SNMP-based real-time measurement systems for IP flows. The system retrieves information from IP packet header it monitors and distinguishes different types of flow information and stores in RTFM MIB tables. Cisco's

NetFlow [8] works in similar ways, but does not use SNMP framework. However, NetFlow system can be integrated with the RTFM manager because RTFM manager can decode NetFlow's data format. Various real-time flows are measured and statistical information can be reported periodically. IETF's RMON [9] and its extension RMON2 [10] are also able to monitor real-time traffic statistics on local area networks within the SNMP framework. However, these systems only collect information from a single network point. The systems are not aware of IP topology or routing protocols. If there is a need to construct a network-wide view of packet transmission, additional effort should be made to the current architecture. Besides, the systems do not provide edge-to-edge performance information.

IETF IP Performance Metrics (IPPM) working group defines a set of standard metrics that can be applied to the quality, performance, and reliability of Internet data delivery services [67]. These metrics are designed such that they can be performed by network operators, end users, or independent testing groups. It is important that the metrics do not represent a value judgment (i.e. define "good" and "bad"), but rather provide unbiased quantitative measures of performance.

7.2 Monitoring End-to-End QoS

There exist several research activities to provide methods for monitoring end-to-end QoS framework. There are two kinds of distinct monitoring methods; active monitoring and passive probing. Active monitoring measures performance of network status by injecting and detecting intentional test traffic at several monitoring points in networks. Passive monitoring measures performance of network status by probing network packets crossing some monitoring points. The passive monitoring does not generate additional traffic on networks and thus does not affect the current network status. However, generally, the active monitoring is more flexible than the passive monitoring because the test traffic can be created with various purposes.

The well-known passive end-to-end monitoring method is PingER project [84]. PingER (Ping End-to-end Reporting) is the name given to the Internet End-to-end Performance Measurement (IEPM) project to monitor end-to-end performance of Internet links. The main mechanism used is the Internet Control Message Protocol (ICMP) Echo mechanism, also known as the Ping facility. This allows you to send a packet of a user selected length to a remote node and have it echoed back. Nowadays it usually comes pre-installed on almost all platforms, so there is nothing to install on the clients. The server (i.e. the echo responder) runs at a high priority (e.g. in the kernel on Unix) and so is more likely to provide a good measure of network performance than a user application. It is very modest in its network bandwidth requirements. PingER uses ping to measure the response time (round trip time in milliseconds (ms)), the packet loss percentages, the variability of the response time both short term (time scale of seconds) and longer, and the lack of reachability, i.e. no response for a succession of pings.

Jiang et al. [11, 35] proposed a distributed passive QoS monitoring mechanism in IP networks. An agent called a 'relevant monitor,' resides at several network points for monitoring real-time flows and reports collected information to monitoring applications. A central database, called a 'real-time application name server,' is used for monitoring applications to locate proper relevant monitors to retrieve specific flow information. The system proposes a distributed mechanism to construct end-to-end flow information. However, it does not understand routing topology and fails in showing how to construct network-wide flow information in detail.

Feldmann et al. [12, 36] have developed the 'NetScope' toolkit that integrates accurate models of topology, traffic, and routing with a flexible visualization environment to support traffic engineering in large ISP networks. The approach provides a network wide view of routing topology with performance statistics on network links. The purpose of the system is to locate heavily-loaded links so that traffic engineering tasks can distribute the load to other possible links. The method is able to determine which network link is overloaded and the NetScope

system has been applied to general IP networks.

7.3 DiffServ Per-Domain Behaviors (PDB)

Recently, Per-Domain Behavior (PDB) has gained more attention than PHB in the IETF DiffServ working group [10, 11]. PDB is defined as the expected treatment that an identifiable group of packets will receive from the 'edge to edge' of a DiffServ domain. While PHB is used to describe forwarding path behaviors required in a router, PDB is used for describing forwarding behaviors across a DiffServ domain. Figure 41 illustrates the relationship of PHB and PDB in a DiffServ domain.

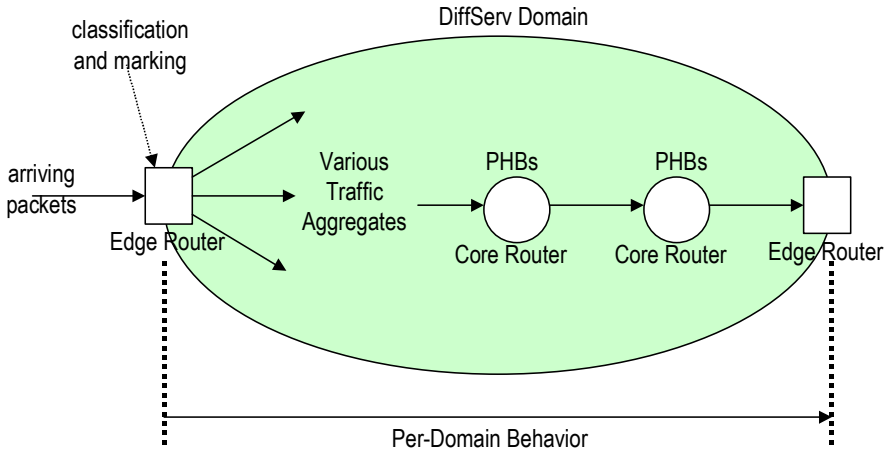


Figure 41. PHB and PDB Relationship in a DiffServ Domain

Using the classification and marking performed at edge routers, arriving packets are mapped to a set of traffic aggregates which are handled by core routers inside the DiffServ domain. Each DiffServ router has a unique method to handle network packets with the implementation of variable resource allocation and queue prioritization. PHB standardizes and unifies these implementation differences among DiffServ routers so that similar predefined behaviors are

performed and observed in every router. Since the useful differentiated services are constructed by basic hop-by-hop resource allocation, standard behaviors are necessary. Further, PHB can be used as a reference when a router chooses and creates local forwarding and shaping mechanisms.

PDB is defined as having the same role as PHB but has extended definitions. When a network service is deployed in the Internet, the end-to-end service connections usually pass through multiple administrative domains. Since DiffServ is deployed and controlled within each administrative domain, every DiffServ domain will have different forwarding polices on incoming packets. However, to control and guarantee service quality over multiple DiffServ domains, each domain should interact and negotiate with each other concerning the forwarding characteristics of service packets. Since PHB is limited to describe behaviors in a router and not appropriate to represent a whole picture of the packet forwarding in a DiffServ domain, an extended approach to describe the behaviors is needed.

The idea of describing network-wide pictures of edge-to-edge traffic aggregates will greatly assist the management of the DiffServ domain. Since every network service is provided over multiple network hops, characteristics of edge-to-edge traffic aggregates can be proper indications for understanding network-wide status of the DiffServ domain. If we can obtain edge-to-edge characteristics of every flow in the DiffServ domain, we can perform useful management operations.

7.4 Management of DiffServ Networks

In order to realize QoS-enabled IP services, it is necessary to have a QoS management framework. A QoS management system following the QoS management framework needs to be designed and developed to manage various QoS components of IP QoS networks.

SNMP-based QoS management is widely accepted as an effective

management solution for IP-based network elements. For each QoS-enabled network device, SNMP MIB is designed and an SNMP agent is installed in each network device. A QoS management system is an SNMP manager that keeps polling MIB values from a set of SNMP agents running in network devices and collects asynchronous TRAP operations from them.

As previously stated, DiffServ network needs to be provisioned before providing services to the end users and applications. Managing DiffServ network needs to provide configuration and provisioning of QoS parameters in each DiffServ router.

SNMP MIB for configuration DiffServ router has been proposed from the IETF DiffServ working group. DiffServ MIB contains various configuration parameters converted to SNMP SMI objects [68]. Agent implementation on the DS MIB can be used as a configuration module in DiffServ routers.

Recently, in order to overcome limitations in flexible configurations of network devices by SNMP methods, policy-based network management architecture has been proposed [15]. Policy Information Base (PIB) is a new management information for configuring network devices by policy-based mechanisms. An alternative approach for configuring DiffServ routers is using COPS protocol with policy-based network management architecture. Policy Information Base (PIB) for DiffServ QoS parameters is also proposed from the IETF DiffServ working group [69]. The DiffServ PIB describes a structure for specifying policy information that can then be transmitted to a network device for the purposes of configuring policy at that device. The model underlying this structure is one of well-defined policy rule classes and instances of these classes residing in a virtual information store called the Policy Information Base (PIB).

After configuring DiffServ routers, it is needed to monitor current network QoS of each DiffServ class the DiffServ routers handle. For this purpose, the RMON working group has been proposed DiffServ RMON (DSMON) MIB [47]. DSMON MIB defines a set of SNMP MIB objects containing traffic statistics gathered in each DiffServ router. Developed with the DSMON MIB, an SNMP

agent is able to collect QoS statistics of each DiffServ class provided in each DiffServ router. Constant polling for the MIB variables can give network administrators a set of valuable statistical trend of each DiffServ service class.

For intradomain management of DiffServ domains neighboring each other, a bandwidth broker architecture has been suggested. Within a DiffServ domain, there exists a bandwidth broker, which is a management system representing the DiffServ domain and negotiates available QoS resources and traffic demands with neighbor domains. When an end-to-end service is established over multiple DiffServ domains, bandwidth brokers from each DiffServ domains coordinate and reserve available network resources for demanded traffic bandwidth. With this architecture, end-to-end QoS can be guaranteed though the request crosses multiple DiffServ domains.

For overall QoS management framework satisfying all the above management requirements, there exist some architectural design researches. One of numbers of major research work is TEQUILA project [25].

8. Conclusions

This chapter summarizes overall contents of the thesis and lists a set of contributions this thesis achieved. In addition, future work that has to be researched further is proposed.

8.1 Summary

A best-effort service model for the Internet is simple and easy to maintain. However, the model does not satisfy various QoS requirements in the Internet, especially when network bandwidth becomes scarce. The Internet is currently facing an urgent need for QoS support from various network users and applications. Differentiated Services (DiffServ) is gaining acceptance as a promising solution towards providing QoS support in the Internet. When DiffServ is deployed in the backbone networks, it is necessary to manage the DiffServ domain by monitoring topology and performance parameters from DiffServ network elements. However, though the operational architecture of DiffServ is becoming mature and stable by the standardization efforts from the IETF DiffServ working group, the management aspect must be refined and extended.

In this thesis, a method for measuring and constructing QoS of edge-to-edge DiffServ flows within a DiffServ domain is proposed. Distributed measurement of QoS information is aggregated by predefined aggregation rules according to topological DiffServ model. By following our methods, the administrators of DiffServ networks can obtain dynamic status of network behaviors very efficiently and also pinpoint bottleneck points and possible solutions when edge-to-edge QoS is not satisfied when congestion occurs.

Two different detailed distributed edge-to-edge throughput monitoring methods have been proposed. The first method provides minimum and maximum

boundary of throughput and packet drops of an edge-to-edge DiffServ flow when ingress / egress pair is specified. The second method provides the exact amount of throughput of the edge-to-edge DiffServ flow. Various applications, such as detailed network status reporting, dynamic adaptive provisioning, and bottleneck detection and resolution, in management of DiffServ domains with the proposed monitoring methods are explained with detailed examples. Main ideas presented in this thesis are depicted in Figure 42.

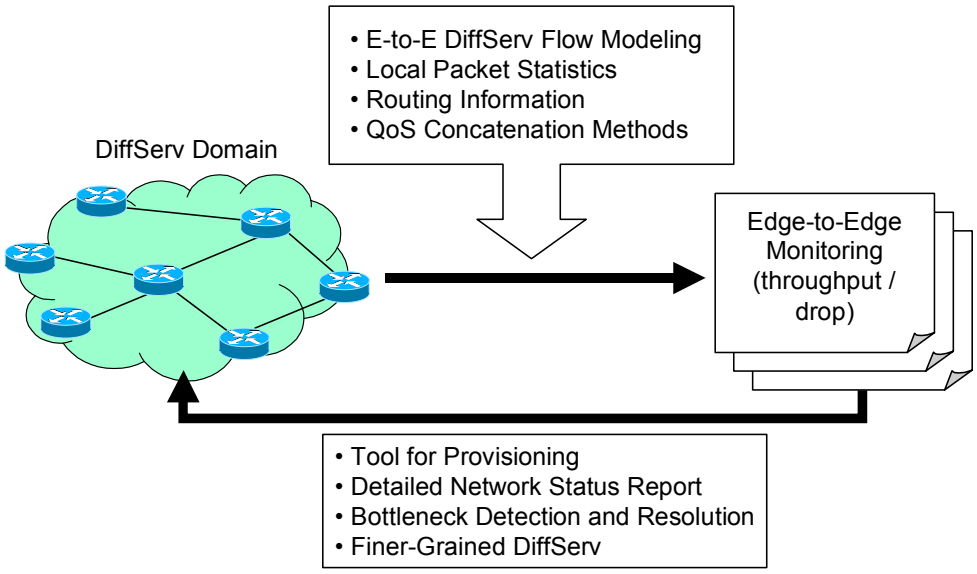


Figure 42. Summary of Main Ideas

Finally, in order to verify and realize the proposed monitoring methods, an SNMP-based edge-to-edge DiffServ flow management system is designed and implemented on a Linux-based testbed. The testbed consists of a set of DiffServ-enabled routers with SNMP agents, a central monitoring manager, and a Java-based console viewer for human administrators. Implementation experiences verifies that the SNMP-based management of DiffServ domains is a simple and effective solution for future QoS-enabled Internet environment.

8.2 Contributions

The primary contribution of this thesis is that the suggested edge-to-edge throughput / drop monitoring methods provide the way of distinguishing aggregated DiffServ traffic flows in a single DiffServ class, which could not be separated in original DiffServ networks. With the proposed edge-to-edge DiffServ flow modeling, DiffServ flows with different source and destination edge routers are considered as different edge-to-edge DiffServ flows to each other. The suggested edge-to-edge monitoring methods successfully detect and de-aggregate the edge-to-edge DiffServ flows from a single DiffServ class of traffic flows.

The suggested QoS monitoring methods increase the granularity of the DiffServ services by considering both routing path and local QoS information. That is, the same DiffServ class in the original DiffServ framework is distinguished by different ingress / egress pairs in the proposed monitoring framework. This enables a DiffServ management system to divide an aggregate traffic flows into a set of different edge-to-edge traffic flows and process them with different treatments. Thus, the service granularity of a DiffServ domain is increased by the proposed framework.

Next, this thesis provides efficient methods that are applicable to various DiffServ management functions. Edge-to-edge throughput monitoring methods can be a useful building block for sophisticated management functions, such as configuration and provisioning, end-to-end traffic performance and QoS monitoring, traffic path control according to dynamic load changes, and accounting and billing for SLA management.

The thesis also provides a model of developing a DiffServ management system using SNMP framework. Developers who wish to create such a system are able to obtain valuable detailed implementation ideas from the thesis. The SNMP agent implementation with DS MIB is one of the initial implementation approaches made so far worldwide.

8.3 Future Work

This thesis proposes two edge-to-edge QoS monitoring methods and shows the applicability of the proposed methods. However, a complete QoS management system needs to be designed and developed in order to effectively use the monitoring results. The complete QoS management system analyzes the collected information and re-configures QoS parameters of all DiffServ devices in a DiffServ domain. The detailed feedback mechanisms and operational scenarios in the complete QoS management system need further research efforts.

Extended high-level service management functionality should be designed and developed. Modeling various SLA and TCS, SLA negotiations, dynamic service management functionality are necessary when deploying a DiffServ management system within network environment where multiple DiffServ domains are interconnected each other.

Finally, since a session of network service is completed when communications in both directions between two edge routers exist, it is necessary to consider DiffServ flows in both directions. Current DiffServ framework treats the two connections independently. However, considering both directions at the same time for enhancing QoS of network services is surely important. The suggested edge-to-edge DiffServ flow modeling and monitoring methods might be extended to support the bi-directional edge-to-edge DiffServ flows.

References

[1] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An Architecture for Differentiated Services," IETF RFC 2475, December 1998.

[2] R. Braden, D. Clark, and S. Shenker, "Integrated Services in the Internet Architecture: an Overview," IETF RFC 1633, June 1994.

[3] K. Nichols, S. Blake, F. Baker and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers," IETF RFC 2474, December 1998.

[4] V. Jacobson, K. Nichols, and K. Poduri, "An Expedited Forwarding PHB," IETF RFC 2598, June 1999.

[5] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski, "Assured Forwarding PHB Group," IETF RFC 2597, June 1999.

[6] N. Brownlee, C. Mills, G. Ruth, "Traffic Flow Measurement: Architecture," IETF RFC 2063, January 1997.

[7] N. Brownlee, "Traffic Flow Measurement: Experiences with NeTraMet," IETF RFC 2123, March 1997.

[8] Cisco NetFlow white paper, http://www.cisco.com/warp/public/cc/pd/iosw/ioft/neflct/tech/napps_wp.htm .

[9] S. Waldbusser, "Remote Network Monitoring Management Information Base," IETF RFC 2819, May 2000.

[10] S. Waldbusser, "Remote Network Monitoring Management Information Base Version 2 Using SMIv2," IETF RFC 2021, January 1997.

[11] Y. Jiang, C. Tham, and C. Ko, "Challenges and Approaches in Providing QoS Monitoring," International Journal of Network Management, October 2000, pp.322-334.

[12] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, and J. Rexford, "NetScope: Traffic Engineering for IP Networks," IEEE Network, Vol.14, No.2, March/April 2000, pp.11-19.

[13] Y. Bernet, D. Durham, and F. Reichmeyer, “Requirements of Diff-serv Boundary Routers,” IETF Internet-Draft, draft-bernet-diffedge-01.txt, November 1998.

[14] Y. Bernet, A. Smith, S. Blake, and D. Grossman, “A Conceptual Model for Diffserv Routers,” IETF Internet-Draft, draft-ietf-diffserv-model-03.txt, May 2000.

[15] D. Durham, J. Boyle, R. Cohen, S. Herzog, R. Rajan, and A. Sastry, “The COPS (Common Open Policy Service) Protocol,” IETF RFC 2748, January 2000.

[16] R. Yavatkar et al., “COPS Usage for Differentiated Services,” IETF Internet-Draft, draft-ietf-rap-cops-pr-00.txt, December 1998.

[17] F. Baker, K. H. Chan, and A. Smith, “Management Information Base for Differentiated Services Architecture,” IETF Internet-Draft, draft-ietf-diffserv-mib-03.txt, May 2000.

[18] W. Stalling, *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2*, 3rd Edition, Addison-Wesley, 1999.

[19] ns-2 network simulator, http://www.isi.edu/nsnam/ns/.

[20] S. N. Bhatti and J. Crowcroft, “QoS-Sensative Flows: Issues in IP Packet Handling,” IEEE Network Computing, Vol.4, No.4, July/August 2000, pp. 48-57.

[21] T. Ma and B. Shi, “Bringing Quality Control to IP QoS,” Network Magazine, November 2000, http://www.networkmagazine.com/article/NMG20001103S0009.

[22] R. Grigonis, “Working the QoS Puzzle,” Computer Telephony, January 2001, http://www.computertelephony.com/article/CTM20001221S0001.

[23] L. Mathy, C. Edwards, and D. Hutchison, “The Internet: A Global Telecommunications Solution?,” IEEE Network, July 2000, pp. 46-57.

[24] B. Carpenter and D. Kandlur, “Diversifying Internet Delivery,” IEEE Spectrum, Vol. 36, No. 11, November 1999, pp.57-61.

[25] P. Trimintzios, I. Andrikopoulos, G. Pavlou, C. F. Cavalcanti, D. Goderis, Y. T’Joens, P. Georgatsos, L. Georgiadis, D. Griffin, R. Egan, C. Jacquenet,

and G. Memenios, “An Architectural Framework for Providing QoS in IP Differentiated Service Networks,” Proc. of 2001 IEEE/IFIP International Symposium on Integrated Network Management, Seattle, Washington USA, 14-18 May 2001, pp. 17-34.

[26] F. Baker, “Requirements for IP Version 4 Routers,” IETF RFC 1812, June 1995.

[27] T. M. Chen, “Network Traffic Measurements and Experiments,” IEEE Communications Magazine, Vol. 38, No. 5, May 2000, pp. 120.

[28] C. Aurrecoechea, A. T. Campbell, and L. Hauw, “A Survey of QoS Architecture,” Multimedia Systems Journal, May 1998, Vol. 6, pp. 138-151.

[29] J.-P. Richter and H. de Meer, “Towards Formal Semantics for QoS Support,” Proc. of the 17th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM’98), 1998, pp. 472 –479.

[30] G. Armitage, *Quality of Service in IP Networks: Foundataion for a Multi-Service Internet*, Macmillan Technical Publishing, April 2000.

[31] P. Ferguson and G. Huston, *Quality of Service: Delivering QoS on the Internet and in Coporate Networks*, Wiley Computer Publishing, 1998.

[32] D. Verma, *Supporting Service Level Agreements on IP Networks*, Macmillan Technical Publishing, 1999.

[33] K. Kilkki, *Differentiated Services for the Internet*, Macmillan Technical Publishing, 1999.

[34] S. Vegesna, *IP Quality of Service*, Cisco Press, 2001.

[35] Y. Jiang, C.-K. Tham, and C.-C. Ko, “A QoS Distribution Monitoring Scheme for Performance Management of Multimedia Networks,” Proc. of 1999 Global Telecommunications Conference (GLOBECOM’99), pp. 64-68.

[36] R. Caceres, N. Duffield, A. Feldmann, J. D. Friedmann, A. Greenberg, R. Greer, T. Johnson, C. R. Kalmanek, B. Krishnamurthy, D. Lavelle, P. P. Mishra, J. Rexford, K. K. Ramakrishnan, F. D. True, and J. E. van der Merwe, “Measurement and Analysis of IP Network Usage and Behavior,” IEEE Communications Magazine, Vol. 38, No. 5, May 2000, pp. 144-151.

[37] J. Postel, “Internet Protocol Specification,” IETF RFC 791, 1981.

[38] B. Davie and Y. Rekhter, *MPLS: Technology and Applications*, Morgan Kaufmann Publishers, 2000.

[39] A. Viswanathan, N. Feldman, Z. Wang, and R. Callon, "Evolution of Multiprotocol Label Switching," IEEE Communications Magazine, Vol. 36, No. 5, May 1998, pp. 165-173.

[40] X. Xiao, A. Hannan, B. Bailey, and L.M. Ni, "Traffic Engineering with MPLS in the Internet," IEEE Network, March 2000, pp. 28-33.

[41] A. Ghanwani, B. Jamoussi, D. Fedyk, P. Ashwood-Smith, L. Li, and N. Feldman, "Traffic Engineering Standards in IP Networks Using MPLS," IEEE Communications Magazine, Vol. 37, No. 12, December 1999, pp. 49-53.

[42] G. Eichler, H. Hussmannm G. Mamais, I. Venieris, C. Prehofer, and S. Salsano, "Implementing Integrated and Differentiated Services for the Internet with ATM Networks: A Practical Approach," IEEE Communications Magazine, Vol. 38, No. 1, January 2000, pp. 132-141.

[43] Y. Bernet, "The Complementary Roles of RSVP and Differentiated Services in the Full-Service QoS Network," IEEE Communications Magazine, Vol. 38, No. 2, February 2000, pp. 154-162.

[44] I. Andrikopoulos and G. Pavlou, "Supporting Differentiated Services in MPLS Networks," Proc. of the 17th International Workshop on Quality of Service (IWQoS '99), 1999, pp. 207-215.

[45] N. Rouhana and E. Horlait, "Differentiated Services and Integrated Services Use of MPLS," Proc. of the 5th IEEE Symposium on Computers and Communications (ISCC 2000), 2000, pp. 194-199.

[46] IETF RMONMIB working group homepage, http://www.ietf.org/html.charters/rmonmib-charter.html.

[47] A. Bierman, "Remote Monitoring MIB Extensions for Differentiated Services," IETF Internet-Draft, draft-ietf-rmonmib-dsmon-mib-05.txt, July 2001.

[48] S. Waldbusser, "Remote Network Monitoring Management Information Base Version 2 Using SMIv2," IETF RFC 2021, January 1997.

[49] Y. Bernet, J. Binder, S. Blake, M. Carlson, B. E. Carpenter, S. Keshav, B. Ohlman, D. Verma, Z. Wang, and W. Weiss, “A Framework for Differentiated Services,” draft-ietf-diffserv-framework-02.txt, Internet Draft, IETF, February 1999.

[50] S. Shenker, C. Partridge, and R. Guerin, “Specification of Guaranteed Quality of Service,” IETF RFC 2212, September 1997.

[51] J. Wroclawski, “Specification of the Controlled-Load Network Element Service,” IETF RFC 2211, September 1997.

[52] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jarmin, “Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification,” IETF RFC 2205, September 1997.

[53] A. Mankin, F. Baker, B. Braden, S. Bradner, M. O’Dell, A. Romanow, A. Weinrib, and L. Zhang, “Resource ReSerVation Protocol (RSVP) Version 1 Applicability Statement Some Guidelines on Deployment,” IETF RFC 2208, September 1997.

[54] J. Wroclawski, “The Use of RSVP with IETF Integrated Services,” IETF RFC 2210, September 1997.

[55] A.K.J. Parekh, “A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks, MIT Laboratory for Information and Decision Systems,” Report LIDS-TH-2089, Laboratory for Informationand Decision Systems, MIT, February 1992.

[56] P. Newman, T. Lyon, and G. Minshall, “Flow Labeled IP: ATM Under IP,” Proceedings of INFOCOM ’96, San Francisco, March 1996, pp.1251-1260.

[57] Y. Katsube, K. Nagami, and H. Esaki, “Toshiba’s RouterArchitecture Extensions for ATM: Overview,” IETF RFC 2098, February 1997.

[58] A. Viswanathan, N. Feldman, and R. Boivie, “ARIS: Aggregate Route-Based IP Switching,” IBM Technical Report TR29.2353, February 1998.

[59] Y. Rekhter, B. Davie, D. Katz, E. Rosen, and G. Swallow, “Cisco System’s Tag Switching Overview,” IETF RFC 2105, February 1997.

[60] A. Acharya, R. Dighe, and F. Ansari, “A Framework for IP Switching Over Fast ATM Cell Transport (IPSOFACTO),” Proceedings of Broadband

Networking Technology Conference, Dallas, TX, Vol.3233, Ch. 36, November 1997, pp.20-28.

[61] E. Rosen, A. Viswanathan, and R. Callon, "Multiprotocol Label Switching Architecture," IETF RFC 3031, January 2001.

[62] L. Andersson, P. Doolan, N. Feldman, A. Fredette, and B. Thomas, "LDP Specification," IETF RFC 3036, January 2001.

[63] Y. Bernet, P. Ford, R. Yavatkar, F. Baker, L. Zhang, M. Speer, R. Braden, B. Davie, J. Wroclawski, and E. Felstaine, "A Framework for Integrated Services Operation over Diffserv Networks," IETF RFC 2998, November 2000.

[64] G. Mamais, M. Markaki, G. Politis, and I. S. Venieris, "Efficient Buffer Management and Scheduling in a Combined IntServ and DiffServ Architecture: A Performance Study," Proc. of the 2nd International Conference on ATM, 1999, pp. 236-242.

[65] A. Detti, M. Listanti, S. Salsano, and L. Veltri, "Supporting RSVP in a Differentiated Service Domain: an Architectural Framework and a Scalability Analysis," Proceedings of ICC '99, May 1999.

[66] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," IEEE/ACM Transactions on Networking, Vol. 1, No. 4, August 1993, pp. 397-413.

[67] V. Paxon, G. Almes, J. Mahdavi, and M. Mathis, "Framework for IP Performance Metrics," IETF RFC 2330, May 1998.

[68] F. Baker, K. Chan, and A. Smith, "Management Information Base for the Differentiated Services Architecture," IETF Internet draft, draft-ietf-diffserv-mib-11.txt, August 2001.

[69] M. Fine, K. McCloghrie, J. Seligson, K. Chan, C. Bell, A. Smith, and F. Reichmeyer, "Differentiated Services Quality of Service Policy Information Base," IETF Internet draft, draft-ietf-diffserv-pib-04.txt, July 2001.

[70] David Perkins and Evan McGinnis, *Understanding SNMP MIBs*, Prentice Hall, 1997.

[71] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser, “Structure of Management Information for Version 2 of the Simple Network Management Protocol (SNMPv2)”, RFC 1902, January 1996.

[72] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser, “Textual Convention for Version 2 of the Simple Network Management Protocol (SNMPv2)”, RFC 1903, January 1996.

[73] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser, “Conformance Statements for Version 2 of the Simple Network Management Protocol (SNMPv2)”, RFC 1904, January 1996.

[74] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser, “Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2)”, RFC 1905, January 1996.

[75] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser, “Transport Mappings for Version 2 of the Simple Network Management Protocol (SNMPv2)”, RFC 1906, January 1996.

[76] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser, “Management Information Base for Version 2 of the Simple Network Management Protocol (SNMPv2)”, RFC 1907, January 1996.

[77] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser, “Coexistence between Version 1 and Version 2 of the Internet-standard Network Management Framework”, RFC 1908, January 1996.

[78] ISO/IEC, “Information Processing – Open Systems Interconnection – Specification of Abstract Syntax Notation One (ASN.1)”, International Standard 8824, 1987.

[79] Uyless Black, *Network Management Standards*, 2nd edition, McGrow Hill, 1994.

[80] LBNL’s Network Research Group homepage, http://www-nrg.ee.lbl.gov/.

[81] TCPdump homepage, http://www.tcpdump.org/.

[82] Ethereal homepage, http://www.ethereal.com/.

[83] Network-Top homepage, http://www.ntop.org/.

[84] PingER project homepage, http://www-iepm.slac.stanford.edu/pinger/.

[85] J. Sadler and B. Mack-Crane, Generalized Switch Management Protocol (GSMP), IETF Internet Draft, draft-sadler-gsmp-tdm-labels-00.txt, February 2001.

[86] H. Sjostrand, J. Buerkle, and B. Srinivasan, Definitions of Managed Objects for the General Switch Management Protocol (GSMP), IETF Internet Draft, draft-ietf-gsmp-mib-05.txt, June 2001.

요 약 문

인터넷 트래픽의 증가와 네트워크 용량의 부족으로 인한 서비스 품질 저하, 일정량의 서비스 품질을 요구하는 새로운 인터넷 서비스의 등장, 서비스 품질에 대한 사용자 및 서비스 제공자의 요구 등에 의해, IP 네트워크 상에서 QoS를 보장하기 위한 여러 연구가 진행되고 있다. 그 중에서도 edge / core 라우터 모델과 소수의 서비스 클래스 분류 및 QoS 보장 방법을 이용하는 Differentiated Services (DiffServ) 방식이 간단한 구조와 좋은 확장성을 가지고 있어서 대규모 인터넷 백본 네트워크에서의 QoS 제공 방식으로 표준화되고 있다. 그러나, DiffServ 방식을 제공하기 위해서는 서비스 제공자가 사용자의 서비스 요구 내용을 바탕으로 해당 네트워크의 자원을 효율적으로 할당하고, 네트워크 변동 상황에 따라 재배치하는 관리 구조가 필수적이다. 이러한 관리 구조 및 세부적인 관리 방식에 대한 연구는 부족한 편이다. 본 학위 논문에서는 DiffServ 네트워크 상에서의 트래픽 플로우를 관리하기 위한 edge 종단간의 분산 QoS 모니터링 구조와 방식을 제안한다. 현재의 DiffServ 네트워크 구조에서는 라우팅 정보를 QoS 제공 방식과 분리하고 있으나, 본 학위 논문에서는 DiffServ 네트워크의 각 edge 종단간 라우팅 정보를 이용하여 edge 종단간의 DiffServ 플로우의 throughput을 모니터링한다. 우선, 모니터링 하고자 하는 edge 종단간의 DiffServ 플로우를 모델링하고, edge 종단간 DiffServ 플로우를 라우팅 경로 상의 라우터가 분산 모니터링한 throughput 정보를 이용하여 구성하는 두 가지 방법을 제안한다. 첫번째 방법은 라우팅 경로상의 트래픽 전송량을 이용하여 edge 종단간의 전송량의 최대 및 최소값을 구하여, 이를 바탕으로 edge 종단간의 throughput과 packet drop 양을 모니터링한다. 두번째 방법은 각 ingress edge 라우터에서 자신의 edge ID를 모든 패킷에 표시하여, 각 egress edge 라우터에서 edge 종단간의 해당 DiffServ 클래스의 트래픽 전송량을 파악하고, 이를 바탕으로 edge 종단간 throughput과 packet drop 양을 모니터링한다. 시뮬레이션을 통해 RED 큐를 사용하는 DiffServ 네트워크 패스 상의 패

킷 손실률은 해당 패스를 지나는 서로 다른 edge 종단간 DiffServ 플로우의 양에 비례한다는 점을 확인하여 제안한 모델링과 edge 종단간 throughput 구성 방식의 타당성을 보이고, 두 가지 방식의 장단점을 비교 분석한다. 또한, 제안한 방식으로 만들어진 정보를 이용하여 DiffServ 네트워크의 초기 설정 작업과 병목 지점 발견과 같은 DiffServ 네트워크 관리에 이용할 수 있음을 보인다. 끝으로, 제안한 분산 throughput 모니터링 및 관리 방식을 리눅스 기반의 DiffServ 테스트 네트워크에 적용하여 SNMP 기반의 edge 종단간 DiffServ 플로우 관리 시스템을 설계하고 구현하는 과정을 보인다. 제안한 모니터링 구조와 방식을 통해 DiffServ 네트워크 상에서 edge 종단간 bandwidth 사용량을 관리할 수 있도록 하여, 보다 세밀한 DiffServ 서비스 클래스를 제공할 수 있으며, 다양한 DiffServ 네트워크 관리 서비스에 응용할 수 있다.

감사의 글

이 력 서

성　　명 : 김 재 영
생년월일 : 1971년 11월 25일
출 생 지 : 부산
주　　소 : 경북 포항시 남구 대잠동 현대홈타운 103-1206

학 력

1990.3 ~ 1994.2 포항공과대학교 전자계산학과 (B.S.)
1994.3 ~ 1996.2 포항공과대학교 전자계산학과 (M.S.)
1998.3 ~ 2002.2 포항공과대학교 컴퓨터공학과 (Ph.D.)

경 력

1996.3 ~ 1998.2 포항공과대학교 학술정보센터 연구원

논문 실적

International Journal Papers

1. Jae-Young Kim and James Won-Ki Hong, “Distributed QoS Monitoring and Edge-to-Edge QoS Aggregation to Manage End-to-End Traffic Flows in IP Networks Supporting Differentiated Services,” Accepted to appear in Journal of Communications and Networks (JCN), Special Issue on Management of New Networking Infrastructure and Services, December 2001. (SCIE)
2. Jae-Young Kim, James Won-Ki Hong, and Tae-Sang Choi, “Managing Edge-to-Edge Traffic Aggregates in Differentiated Services Networks,” Journal of Networks and Systems Management (JNSM), Special Issue on IP Operations and Management, Vol. 9, No. 3, September 2001, pp. 266-291.

3. Jae-Young Kim, Hong-Taek Ju and James Won-Ki Hong, “Towards TMN-based Integrated Network Management Using CORBA and Java Technologies,” IEICE Transactions on Communications, Special Issue on New Paradigms in Network Management, Vol. E82-B, No.11, November 1999, pp.1729-1741. (SCI)

4. James Won-Ki Hong, Soon-Sun Kwon and Jae-Young Kim, “WebTrafMon: Web-based Internet/Intranet Network Traffic Monitoring and Analysis System,” Computer Communications, Elsevier Science, Vol. 22, No. 14, September 1999, pp. 1333-1342. (SCIE)

5. James Won-Ki Hong, Young-Mi Shin, Myung-Seop Kim, Jae-Young Kim and Young-Ho Suh, “Design and Implementation of a Distributed Multimedia Collaborative Environment,” Cluster Computing, Baltzer Science, Vol. 2, No. 1, January, 1999, pp. 45-49.

<u>International Conference Papers</u>

6. Sook-Hyun Ryu, Jae-Young Kim, and James W. Hong, “Approaches to Support Differentiated Quality of Web Service,” Proc. of QofIS 2001 Conference, September 2001, Coimbra, Portugal, pp. 273-285.

7. Jae-Young Kim, James Won-Ki Hong, and Tae-Sang Choi, “Constructing End-to-End Traffic Flows for Managing Differentiated Services Networks,” Proc. of the 11th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM 2000), December 4-6, 2000, Austin, Texas, USA, pp. 83-94.

8. Sook-Hyang Kim, Jae-Young Kim and James Won-Ki Hong, “A Statistical, Batch, Proxy-Side Web Prefetching Scheme for Efficient Internet Bandwidth Usage,” Proc. of the 2000 Networld+Interop Engineers Conference, Las Vegas, May 2000.

9. Jae-Young Kim, Myung-Sup Kim and James Won-Ki Hong, “Management of Differentiated Services Using the SNMP Framework,” Proc. of the 2nd International Conference on Advanced Communication Technology (ICACT 2000), Muju, Korea, February 16-18, 2000, pp.624-629.

10. Sook-Hyang Kim, Jae-Young Kim and James Won-Ki Hong, “A Statistical

Prefetching Web Caching Scheme for Efficient Internet Bandwidth Usage," Proc. of the 2nd International Conference on Advanced Communication Technology (ICACT 2000), Muju Korea, February 16-18, 2000, pp.137-141.

11. Jae-Young Kim and James Won-Ki Hong, "Design and Implementation of a Web-based Internet/Intranet Mail Server Management System," Proc. of the 1999 IEEE International Conference on Communications (ICC'99), June 6-10, 1999, Vancouver, Canada, pp.641-645.

12. Jae-Young Kim, Young-Min Kang, Seoung-Duck Lim and James Won-Ki Hong, "TMN-based Integrated Network Management Using Web and CORBA Technologies," Proc. of the 6th IFIP/IEEE Symposium on Integrated Network Management (IM'99), May 10-14, 1999, Boston, MA, USA, pp.947-948.

13. Sook-Kyoung Ahn, Mi-Jeong Choi, Jae-Young Kim, James Won-Ki Hong, Sang-Ki Kim, "TMN-based Intelligent Network Management: Local Number Portability," Proc. of the 6th IFIP/IEEE Symposium on Integrated Network Management (IM'99), May 10-14, 1999, Boston, MA, USA, pp 83-96.

14. Jae-Young Kim, Seoung-Duck Lim, and James Won-Ki Hong, "TMN-based Network Management Using Web and CORBA," Proc. of the 1st International Conference on Advanced Communication Technology (ICACT'99), February 10-12, 1999, Muju, Korea, pp.326-331.

15. Soon-Sun Kwon, Jae-Young Kim, and James Won-Ki Hong, "Web-based Internet/Intranet Network Traffic Monitoring and Analysis," Proc. of the 1st International Conference on Advanced Communication Technology, Muju, Korea, February 1999, pp 49-54.

Domestic Journal Papers

16. Jae-Young Kim, and James Won-Ki Hong, "Distributed Edge-to-Edge QoS Measurement and Aggregation in Differentiated Services Networks," KNOM Review, Vol.4, No.1, 2001, pp.48-55.

17. 최연숙, 김재영, 홍원기, "웹 기반의 실시간 인터넷 트래픽 흐름 측정 및 분석," KNOM Review, Vol. 3, No. 1, June 2000, pp. 1-11.

18. Jae-Young Kim and James Won-Ki Hong, "Web-Based Management System for Internet Electronic Mail Service," KNOM Review, Vol.2, No.1, April 1999,

pp.26-33.

19. 권순선, 김재영, 홍원기, "웹 기반의 인터넷/인트라넷 네트워크 트래픽 모니터링", KNOM Review, Vol.2, No.1, April 1999, pp.1-10.

Domestic Conference Papers

20. Jae-Young Kim, and James Won-Ki Hong, “Distributed Edge-to-Edge QoS Measurement and Aggregation in Differentiated Services Networks,” Proc. of KNOM 2001 Conference, Daejeon, May 24-25, 2001, pp.49-54.

21. 홍순화, 김재영, 조범래, 홍원기, “분산 시스템 환경에서의 로드 밸런싱을 통한 웹기반 네트워크 트래픽 모니터링 및 분석,” Proc. of KNOM 2001 Conference, Daejeon, Korea, May 24-25, 2001, pp. 198-205.

22. Jae-Young Kim, James Won-Ki Hong, and Tae-Sang Choi, “Traffic Flow Management in Differentiated Services Networks,” Proc. of KNOM 2000 Conference, Daejeon, May 18-19, 2000, pp. 150-155.