Distributed QoS Monitoring and Edge-to-Edge QoS Aggregation to Manage End-to-End Traffic Flows in Differentiated Services Networks

Jae-Young Kim and James Won-Ki Hong

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 packets 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 domain. The DiffServ framework defines simple and efficient QoS differentiation mechanisms for the Internet. However, the original DiffServ concept does not provide a complete QoS management framework. Since traffic flows in IP networks are unidirectional from one network point to the other and routing paths and traffic demand get dynamically altered, it is important to monitor end-to-end traffic status, as well as traffic status in a single node. This paper suggests a distributed QoS monitoring method that collects the statistical data of each service class in every DiffServ router and calculates edge-to-edge QoS of the aggregated IP flows by combining routing topology and traffic status. A formal modeling of edge-to-edge DiffServ flows and algorithms for aggregating edge-to-edge QoS is presented. Also an SNMP-based QoS management prototype system for DiffServ networks is presented, which validates our QoS management framework and demonstrates useful service management functionality.**

Index Terms: **Differentiated services, QoS monitoring, end-to-end QoS, IP, management, SNMP-based QoS management.**

I. 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, the amount of 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 such points where network bandwidth becomes scarce. Unexpected packet loss, delay, and jitter occur when many network packets compete for insufficient network bandwidths 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 the natural requirement of every user.

In order to provide service differentiation in 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 and thus is very difficult to deploy in large backbone networks. Instead, DiffServ controls the aggregation of traffic flows; that is, DiffServ flows is applied 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.

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. And the edge routers perform marking the most-significant 6 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 the proposed PHBs from the IETF.

However, current DiffServ specifications have limitations in providing complete QoS management framework. As described in Fig. 1, QoS management tasks are composed of repeated cycles of configuration, monitoring, and analysis. Without any one component of the cycle, the QoS management cannot be provided completely. According to this point of QoS management view, current DiffServ RFCs and drafts are mainly for QoS provisioning and configuration only. QoS monitoring and analysis based on the measured QoS parameters are not yet addressed in detail.

In this paper, we provide a distributed QoS monitoring and

Manuscript received July 22, 2001.

J. Y. Kim and J. W. K. Hong are with Department of Computer Science and Engineering Pohang University of Science and Technology, e-mail: jay@postech.ac.kr, jwkhong@postech.ac.kr.

The authors would like to thank the Ministry of Education of Korea for its financial support toward the Electrical and Computer Engineering Division at POSTECH through its BK21 program.

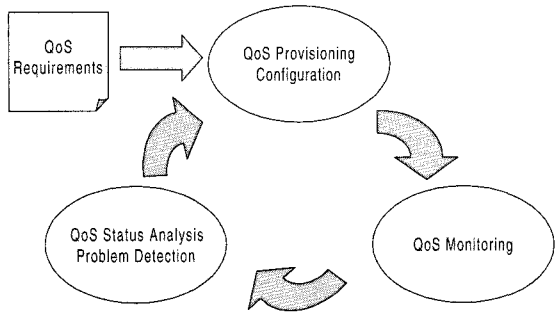


Fig. 1. QoS management cycle.

edge-to-edge QoS aggregation methods in DiffServ networks. Edge-to-edge QoS information is very useful 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 network-wide traffic status in detail. Locating overloaded links and suggesting possible solutions can be drawn from the QoS status of DiffServ flows. In our methods, each DiffServ router in a DiffServ domain monitors QoS information of multiple QoS classes for every network interface. The distributed monitoring has benefits in using less network resources and providing more efficient solution.

The rest of this paper is organized as follows. Section II, investigates related work and Section III, presents the modeling of DiffServ networks and edge-to-edge DiffServ flows. Section IV, describes the suggested distributed edge-to-edge QoS monitoring and aggregation methods with detailed examples. In Section V, an edge-to-edge DiffServ flow management prototype system is designed and implemented to realize our QoS management framework. Finally, Section VI, summarizes our work and discusses possible future work.

II. RELATED WORK

Network QoS monitoring plays an important role when building complete network QoS management systems. We note several related research work concerning QoS monitoring of IP networks.

IETF's real-time flow measurement (RTFM) architecture [6], [7] and Cisco's NetFlow [8] suggest and implement real-time measurement systems for IP flows. The systems retrieve information from IP packet header they monitor and distinguish different types of flow information. 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 must be made to the current architecture. Besides, the systems do not provide edge-to-edge performance information.

Jiang *et al.* [11] proposed a distributed 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 to show how to construct network-wide flow information in detail.

Feldmann *et al.* [12] have developed the 'NetScope' toolkit that integrates models of topology, traffic, and routing with a visualization environment to support traffic engineering in ISP networks. The approach is similar to our work in providing 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. However, the purpose of our method is to show edge-to-edge QoS information for each DiffServ flow. Both methods are able to determine which network link is overloaded. However, only our method is able to determine where the loaded traffic comes from and where it sinks to because our system has QoS information of every edge-to-edge DiffServ flow. Further, the NetScope system has been applied to general IP networks while our method has been applied to QoS-enabled DiffServ networks. Our method is able to differentiate various traffic classes and monitor different QoS parameters.

Our approach is unique in providing edge-to-edge QoS information in QoS-enabled DiffServ networks. The proposed method combines topology and performance information to construct topology-aware QoS information that is helpful in understanding and analyzing QoS provisioning status of the network and managing the DiffServ domain as a whole.

III. MODELING EDGE-TO-EDGE DIFFSERV FLOWS

For modeling edge-to-edge DiffServ flows, we investigate and conceptualize DiffServ components building a DiffServ domain. First, the conceptual model of a DiffServ router is presented and mathematical definitions of topology and flow are given.

A. Conceptual Model of a Diffserv Router and a TCB

A DiffServ domain consists of a set of DiffServ routers under a set of consistent provisioning rules. There are two kinds 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 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 that are organized as in Fig. 2.

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

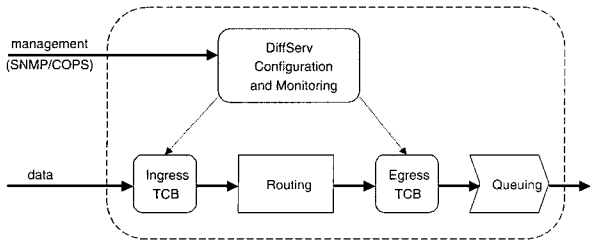


Fig. 2. Conceptual model of a DiffServ router.

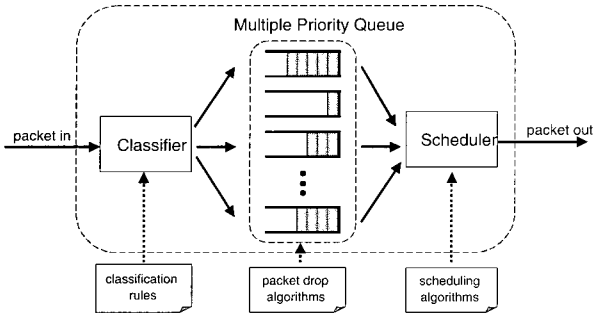


Fig. 3. Conceptual model of a TCB.

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 conceptual TCB can be modeled as in Fig. 3. 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 according to a scheduling algorithm. Traffic flows following the procedures are shaped and tailored to meet the requirements of the given traffic class.

The important thing to consider from this DiffServ router model is that 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, and number of packet drops.

B. Mathematical Modeling of an Edge-to-Edge Diffserv Flow

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 an edge-to-edge DiffServ flow is built on a routing path from a source edge router to a destination edge router. We denote the topology of a DiffServ domain as a set of DiffServ routers (r_i) interconnected with unidirectional routing path. A unidirectional link (l_i) exists

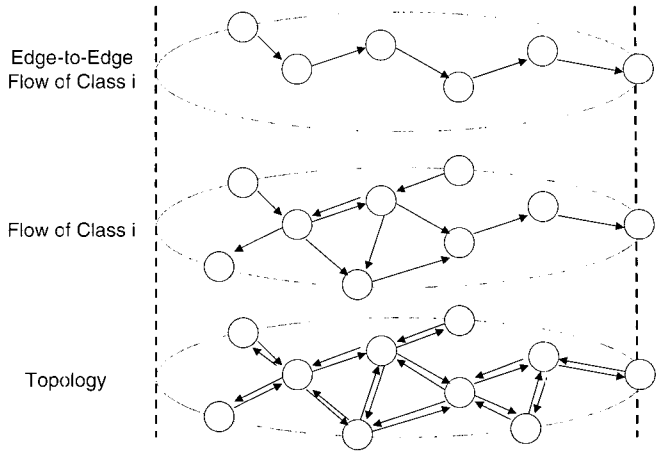


Fig. 4. Mapping relationship of topology, flow, and edge-to-edge flow.

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.

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

In addition to the topological modeling, the dynamic traffic information should be modeled as well. At a certain moment, there is a traffic flow between a set of edge routers within a DiffServ domain. We denote this 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.

$$\mathrm{D}\,(c_i) = (T_i, Q_i), \text{where } T_i = R_i \cup L_i \subseteq T$$
$$Q_i = \{Q(c_i, r_j) | r_j \in R_i\}.$$

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. And Q_i is a set of QoS monitoring information experienced in every router in R_i.

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 route r_d can be denoted as $D(c_i, r_s, r_d)$ and represented as the following equation.

$$D(c_i, r_s, r_d) = (T_{i,r_s,r_d}, Q_{i,r_s,r_d}),$$
$$\text{where } T_{i,r_s,r_d} = R_{i,r_s,r_d} \cup L_{i,r_s,r_d} \subseteq T_i$$
$$Q_{i,r_s,r_d} = \{Q(c_i, r_j) | r_j \in R_{i,r_s,r_d}\} \subseteq Q_i.$$

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. And Q_{i,r_s,r_d} is a set of QoS monitoring information experienced in every router in R_{i,r_s,r_d}, and a subset of Q_i.

Fig. 4 depicts mapping relationships of three hierarchical models. The topology model contains a set of DiffServ flows of different DiffServ classes, and the DiffServ flow model contains a set of different source/destination edge-to-edge DiffServ

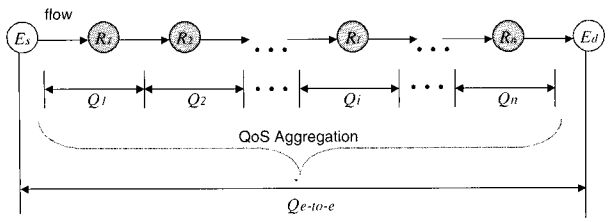


Fig. 5. Distributed QoS measurement and aggregation.

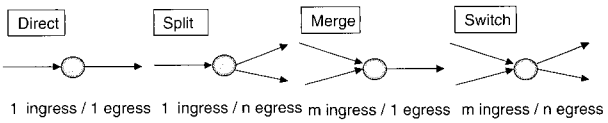


Fig. 6. Graphical representation of DiffServ nodes.

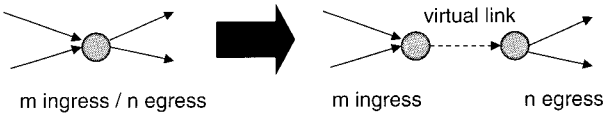


Fig. 7. Conversion of a switch node.

flows. Fig. 4 reveals how to extract an edge-to-edge DiffServ flow from a topology of a DiffServ domain.

When managing DiffServ networks, it is useful 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. The topology, status, and performance of a 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 using 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.

IV. DISTRIBUTED QoS MONITORING AND EDGE-TO-EDGE QoS AGGREGATION

In this section, we propose a distributed QoS monitoring method based on the mathematical modeling of edge-to-edge DiffServ flows presented in the previous section. A graphical representation of an edge-to-edge DiffServ flow is devised and edge-to-edge QoS aggregation rules are explained.

A. Distributed Qos Monitoring and Aggregation

When managing edge-to-edge QoS of a DiffServ flow, we can use two different approaches. One is monitoring QoS at each edge point, and the other is monitoring QoS in every routing point on routing path. Our approach is a distributed monitoring of QoS information at every transit router between two edge routers. And the locally measured QoS information is aggregated by predefined aggregation rules. The distributed QoS monitoring and aggregation method is illustrated in Fig. 5.

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.

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.

B. Graphical Representation of an Edge-to-Edge Diffserv Flow

To represent edge-to-edge DiffServ flows more efficiently, we devised a graphical notation of topology. We show basic graphical notations of DiffServ flows to understand the topological information of each DiffServ flow.

Fig. 6 illustrates four different kinds of DiffServ node representations, composed of a vertex and directed edges.

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 we classify different types of DiffServ nodes is that different kinds of network nodes require different parameter aggregation rules.

Without the loss of generality, the switch node can be converted to a linkage of one merge node and one split node as in Fig. 7. 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.

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

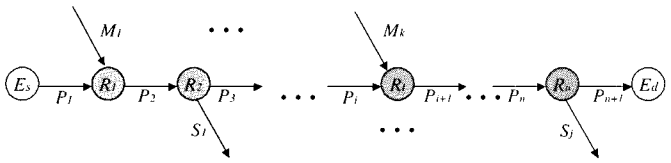


Fig. 8. Graphical repreentation of an edge-to-edge DiffServ flow.

Table 1. Aggregation rules for different parameter/node combination.

	direct	split	merge
throughput	find_min	find_next_hop find_min	calculate_ratio find_min
number of dropped packets	add	find_next_hop add	calculate_ratio add

named 'path link,' a link belonging to a split node and out of the routing path is named 'split link,' and a link beloning 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 Fig. 8.

The directed graph in Fig. 8 is a flow network [17] 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.

C. Edge-to-Edge QoS Aggregation Rules

Locally-monitored performance information and the edge-to-edge routing path are used to generate edge-to-edge QoS of DiffServ flows. We propose a simple hop-by-hop QoS aggregation rules for aggregating QoS values of parameters monitored at each DiffServ node in edge-to-edge routing path. The purpose of the aggregation rules is to extract status of an edge-to-edge DiffServ flow from locally-observed traffic statistics.

Table 1 summarizes the different aggregation rules for each parameter and node combination. Two performance parameters, throughput and number of dropped packets, are explained as examples.

There are four different operations for building up aggregation rules. One aggregation rule consists of one or more individual operations according to the kind of parameter and node combination. A `find_min` operation finds a smaller value between the ingress parameter and the egress parameter. Since the edge-to-edge throughput of a DiffServ flow is bounded by the link of the minimum throughput, a `find_min` operation is used to extract the throughput parameter. An add operation adds the egress parameter to the ingress parameter, so that the edge-to-edge parameter can be accumulated to the end of routing path. The number of dropped packets can be calculated by summing

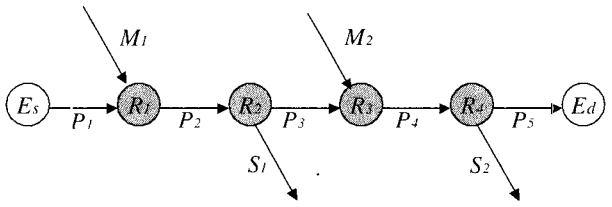


Fig. 9. Example Topology of an Edge-to-Edge DiffServ Flow.

up all the values on the routing path. For the nodes with multiple egress links, such as split nodes, a `find_next_hop` operation must be performed before other operations to specify one egress link that the edge-to-edge DiffServ flow of concern actually uses for the next routing path. For the nodes with multiple ingress links, such as merge nodes, a `calculate_ratio` operation calculates the fraction of the ingress traffic occupied in the egress traffic. The calculation can be represented by the amount of current throughput of DiffServ flow in the ingress link divided by the throughput of the next hop egress link. The following equation provides the value of the fraction from the operation.

$$\text{bandwidth sharing ratio (BSR)} = \frac{\text{current throughput of DiffServ flow in the ingress link}}{\text{through put of the next hop egress link.}}$$

The `calculate_ratio` operation 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 result of the `calculate_ratio` operation 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.

Two detailed examples (one with no packet drop and one with packet drops) are illustrated with the routing topology in Fig. 9. An edge-to-edge DiffServ flow starts at a source edge router Es and sinks at a destination edge router E_s. 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.

The first example case is in the situation that all links have no packet drops. There is no congestion point in this situation and packets injected to the network are completely transferred

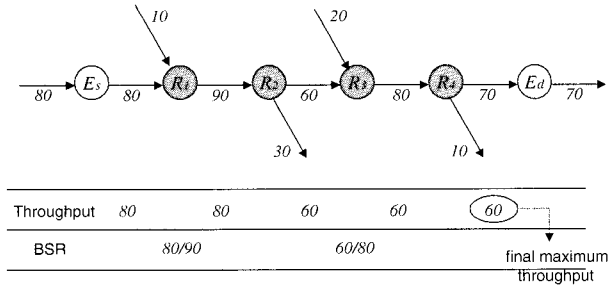


Fig. 10. Example of edge-to-edge aggregation rules (no drop situation).

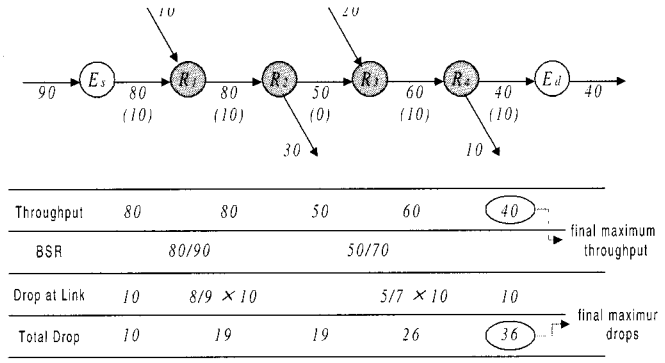


Fig. 11. Example of edge-to-edge aggregation rules (drop situation).

to the desired destination node. Example QoS monitoring information and edge-to-edge aggregation in this situation are given in Fig. 10. The number at each link represents a relative value of the measured bandwidth within a monitoring interval at the link.

At first, the initial throughput received by the source edge router is 80. By following the routing path and the aggregation 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. At two merge nodes, 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 Fig. 11 is in the situation that there are congestion points where available bandwidth is less than the traffic demand and thus packets are dropped. 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 initial amount of traffic received by the soruce edge router is 90. This traffic is controlled for the admission in the source edge router with the packet drop of 10, and only throughput of 80 is transferred to the DiffServ domain. The next node, R_1, is a merge node and receives merged traffic of 10, which is added to the current throughput of 80. Since R_1 is a merge node, the BSR value is calculated as 80/90 because the merged total throughput is 80 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 the 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 BSR value to the current amount of dropped packets and resulted in the value of 8.89 and it is rounded to 9. The total amount of dropped packets in the edge-to-edge DiffServ flow is accumulated to 19 at path P_2.

The maximum throughput is calculated by following the same steps as in the first example and the final value is 40. Because of the packet drops following the routing path, the last link bounds the maximum throughput. As to the amount of packet drops, the aggregation rule at merge node requires to calculate the share ratio and packet drops at the link should be multiplied by the BSR with the amount of packet drops to remove the effect of merged traffic. The final maximum packet drops is calculated to 36.

Table 2. 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

V. DEVELOPING AN EDGE-TO-EDGE DIFFSERV FLOW MANAGEMENT SYSTEM

We have developed an SNMP-based edge-to-edge DiffServ flow management system based on the proposed monitoring methods. 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.

A. 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 is the *de facto* standard for managing Internet-related systems. In this section we explain 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 [18]. 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 2 summarizes the primary object tables defined in the latest DiffServ MIB.

The DiffServ MIB tables are categorized in four architectural DiffServ elements; classifier, meter, action, and queue. Each

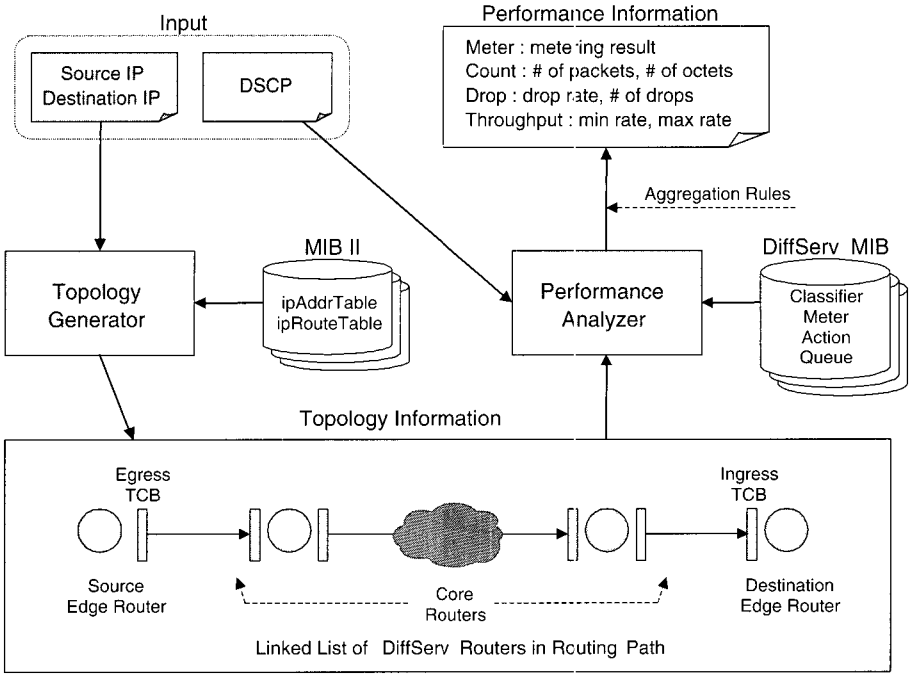


Fig. 12. Construction of edge-to-edge DiffServ flows.

logical element contains several MIB tables and specific MIB objects for describing TCBs in a DiffServ router. The shaded MIB tables in Table 2 are subordinate tables that belong to 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.

B. 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 [18] values in each DiffServ router. Constructing edge-to-edge DiffServ flows thus consists of two phases, as shown in Fig. 12. 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 [19] and DiffServ MIB are used to construct the 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 that has 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.

C. 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 [20]. The

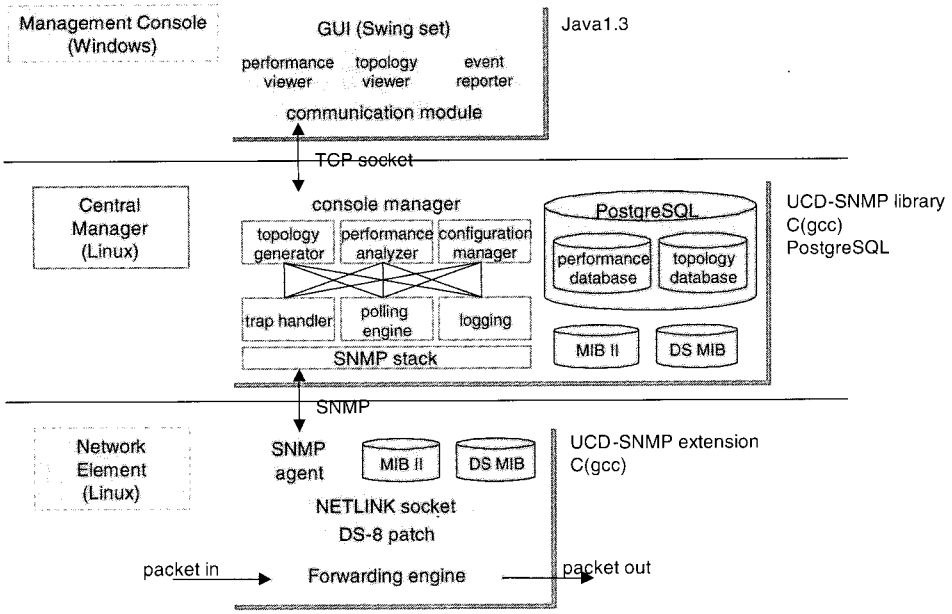


Fig. 13. Implementation architecture of DiffServ domain management system.

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 [21]. The extended DiffServ features are freely available in the form of a kernel patch. The latest DiffServ package (DS-8) available from 'DiffServ on Linux' Web site [22] 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.

We have established a set of DiffServ routers in the Linux systems and have 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. Fig. 13 shows the implementation architecture of our DiffServ domain manager.

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 [23] 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 [24]. 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 we 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 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 [25]. 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.

VI. CONCLUSION AND FUTURE WORK

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, managing 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 paper, we have proposed a method for measuring and constructing QoS of edge-to-edge DiffServ flows within a DiffServ domain. 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 due to congestion.

To verify the suggested methods, we are building a testbed environment with several DiffServ routers configured in Linux platform. Experiments for performance evaluation will be performed on the testbed environment to compare the suggested methods with existing methods. We are also simulating more complicated topology and traffic patterns by using ns-2 network simulator [26]. The experiment and simulation results will provide more helpful information to understand our method and applicability.

A systematic method for representing the proposed DiffServ traffic aggregate information is needed. The proposed construction process and graphical notation must be extended to produce a formal and complete description of the edge-to-edge traffic aggregates. Standardized data formats and extended graphical representations are under research. Further more, mathematical formulation on hop-by-hop aggregation for edge-to-edge traffic aggregates requires much more research, especially concerning multiple source and destination edge routers.

REFERENCES

[1] S. Blake *et al.*, "An architecture for differentiated services," IETF, RFC 2475, Dec. 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 *et al.*, "Definition of the differentiated services field (DS Field) in the IPv4 and IPv6 headers," IETF, RFC 2474, Dec. 1998.

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

[5] J. Heinanen *et al.*, "Assured forwarding PHB group," IETF, RFC 2597, June 1999.

[6] N. Brownlee, C. Mills, and G. Ruth, "Traffic flow measurement: Architecture," IETF, RFC 2063, Jan. 1997.

[7] N. Brownlee, "Traffic flow measurement: Experiences with NeTraMet," IETF, RFC 2123, Mar. 1997.

[8] Cisco NetFlow white paper. Available at 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, Jan. 1997.

[11] Y. Jiang, C. Tham, and C. Ko, "Challenges and approaches in providing QoS monitoring," *Int. J. Network Management*, pp. 322–334, Oct. 2000.

[12] A. Feldmann *et al.*, "NetScope: Traffic engineering for IP networks," *IEEE Network*, vol. 14, no. 2, pp. 11–19. Mar./Apr. 2000.

[13] Y. Bernet, D. Durham, and F. Reichmeyer, "Requirements of Diff-serv boundary routers," IETF, Internet-Draft, draft-bernet-diffedge-01.txt, Nov. 1998.

[14] Y. Bernet *et al.*, "A conceptual model for Diffserv routers," IETF, Internet-Draft, draft-ietf-diffserv-model-03.txt, May 2000.

[15] D. Durham *et al.*, "The COPS (Common Open Policy Service) protocol," IETF, RFC 2748, Jan. 2000.

[16] R. Yavatkar *et al.*, "COPS usage for differentiated services," IETF, Internet-Draft, draft-ietf-rap-cops-pr-00.txt, Dec. 1998.

[17] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, McGraw-Hill, 1990.

[18] F. Baker, K. H. Chan, and A. Smith, "Management information base for differentiated services architecture," IETF, Internet-Draft, draft-ietf-diffserv-mib-12.txt, Sept. 2001.

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

[20] S. Radhakrishnan, "Linux-advanced networking overview-version 1," a technical paper of Department of Electrical Engineering and Computer Science, Univ. of Kansas, Aug. 22, 1999.

[21] W. Almesberger, J. H. Salim, and A. Kuznetsov, "Differentiated services on linux," IETF, Internet-Draft, draft-almesberger-wajhak-diffserv-linux-01.txt, June 1999.

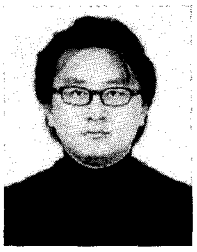
[22] W. Almesberger, "Differentiated services on linux." Available at http://lrcwww.epfl.ch/linux-diffserv/.

[23] UCD-SNMP. Available at http://ucd-snmp.ucdavis.edu/.

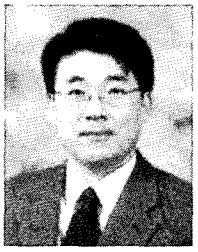
[24] PostgreSQL. Available at http://www.postgresql.org/.

[25] G. Dhandapani and A. Sundaresan, "Netlink Sockets-Overview," Technical paper of Department of Electrical Engineering and Computer Science, University of Kansas, Sept. 13, 1999.

[26] ns-2 network simulator. Available at http://www.isi.edu/nsnam/ns/.



Jae-Young Kim received BSc and MSc degrees from the Pohang University of Science and Technology (POSTECH) in 1994 and 1996, respectively. From 1996 to 1997 he has been a researcher in Computing Center of the same University and worked in intelligent campus and digital library projects. From 1998 he is pursuing his Ph.D. degree in the department of computer science and engineering, POSTECH, Pohang, Korea. His area of research interests includes differentiated services network, distributed computing, and Internet traffic management. He is a student member of IEEE.



James Won-Ki Hong is an associate professor in the Dept. of Computer Science and Engineering, POSTECH, Pohang, Korea. He received a Ph.D. degree from the University of Waterloo, Canada in 1991 and an M.S. degree from the University of Western Ontario in 1985. He has worked on various research projects on network and systems management, with a special interest in Web, Java, and CORBA technologies. His research interests include network and systems management, distributed computing, and multimedia systems. He has published more than 100 international journal and conference papers. He has served as Technical Chair for IEEE CNOM from 1998 to 2000. He was technical co-chair of NOMS 2000 and APNOMS'99. He is a member of IEEE, KICS, KNOM, and KISS.