Journal of Network and Systems Management, Vol. 9, No. 3, September 2001 (© 2001)

Monitoring Edge-to-Edge Traffic Aggregates in Differentiated Services Networks

Jae-Young Kim,[1] James Won-Ki Hong,[1] and Tae-Sang Choi[2]

Differentiated Services (DiffServ), which are currently being standardized in the IETF DiffServ working group, is a solution that can provide different qualities of service to different network users. DiffServ aggregates network packets at edge routers and forwards the aggregated packets to core routers with different priorities. In this paper, we propose methods using the SNMP framework for monitoring edge-to-edge traffic aggregates in a DiffServ domain, which consists of a set of DiffServ-enabled routers. In order to manage each DiffServ router, we have analyzed the DiffServ MIB and instrumented it in the router. Further, we propose monitoring behaviors of edge-to-edge traffic aggregates by combining topology and performance information from MIB II and DiffServ MIB. Construction procedures and graphical representation of the edge-to-edge traffic aggregates are explained in detail. We also extend our efforts to implement a DiffServ domain monitoring system that monitors a set of DiffServ-enabled routers and traffic aggregates between every edge router pair. We believe that the proposed monitoring methods can serve as useful building blocks for managing DiffServ networks.

KEY WORDS: Network Management; Internet QoS Management; SNMP Manager and Agent; End-to-End QoS Monitoring.

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, the amount of network bandwidth is often 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 net-

[1] Department of Computer Science and Engineering, Pohang University of Science and Technology. E-mail: {*jay,jwkhong*}*@postech.ac.kr*

[2] Internet Architecture Team, Internet Technology Department, Electronics and Telecommunications Research Institute. E-mail: *choits@etri.re.kr*

1064-7570/01/0900-0267$19.50/0

work packets compete for insufficient network bandwidths at bottleneck points. Since both people and applications are becoming increasingly dependent on network services, guaranteeing a sufficient amount of service quality has become the natural requirement of every user. Recent network applications have become more time and bandwidth sensitive, and users of network applications are willing to pay more to guarantee their quality-of-service (QoS).

Guaranteeing QoS in the current Internet environment is an onerous task because the Internet is originally designed to provide best-effort service quality, so that every packet being forwarded through the Internet will be treated in the same manner. Both the Transmission Control Protocol (TCP) and the Internet Protocol (IP), two of the most widely used network protocols for delivering packets in the Internet, handle all packets with the same priority and in a first-come-first-serve manner. This best-effort service stems from a clear design policy to sacrifice everything possible for the sake of simplicity. Simplicity enables the success of the Internet, yet creates difficulties in enabling the Internet to accept QoS architecture [1].

To provide service differentiation in the Internet, two different mechanisms, Integrated Services (IntServ) [2] and Differentiated Services (DiffServ) [3], have been introduced by the Internet Engineering Task Force (IETF). IntServ provides three different classes of per-flow service by using Resource reSerVation Protocol (RSVP) [4] signaling. A flow is defined by a 5-tuple that consists of source and destination IP addresses, type of transport protocol, and source and destination application port numbers. DiffServ is an alternative approach to IntServ 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, traffic aggregates, at each routing decision point without using a signaling protocol. A Type of Service (ToS) field is used for distinguishing the traffic aggregates. It is easier to implement DiffServ than IntServ because ToS is much simpler than the 5-tuple flow information. Since DiffServ is a simpler and more scalable solution for Internet backbone networks, it is widely accepted as a feasible solution for providing Internet QoS.

DiffServ applies administrative domain concepts. At the boundary of the DiffServ domain are edge routers marking the ToS fields of incoming packets. The edge routers perform classification of flows based on 5-tuple information like RSVP/IntServ. Within one domain, core routers forward traffic according to the ToS field of the traffic aggregates. Since the edge routers have already marked the ToS field 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 ToS information. The basic architecture of DiffServ is explained further in Section 2.

The operational aspects of the DiffServ networks have been standardized

by the IETF DiffServ working group. However, the management aspects of the DiffServ networks are not yet fully developed. Traditionally, Internet-related systems are managed by the Simple Network Management Protocol (SNMP). Current efforts by the working group include DiffServ Management Information Base (MIB) defined by the SNMP Structure of Management Information (SMI) [5]. However, the management approach is limited to managing a single router itself.

To manage a DiffServ domain, it is obviously necessary to retrieve edge-to-edge traffic status. Since a DiffServ domain has a set of DiffServ routers and a set of traffic classes, there are many edge-to-edge traffic flows at a certain time. For example, if a DiffServ network provides Video-on-Demand (VOD) service from a VOD server to a set of VOD clients, several DiffServ traffic flows may occur from the edge router of the VOD server to several edge routers with VOD clients in their boundary. In DiffServ networks, traffic from one host to another are aggregated in an edge router with the same ToS value. However, current management efforts do not provide such information from DiffServ networks.

In this paper, we propose a mechanism to construct the properties of edge-to-edge traffic aggregates and a system to monitor the status of traffic aggregates. We construct edge-to-edge traffic aggregates that flow through the DiffServ domain by combining topology and performance information from MIB II and DiffServ MIB. Monitoring edge-to-edge traffic aggregates in a DiffServ domain helps administrators to understand the current service status of the DiffServ domain and can be a useful building block to support sophisticated management operations, such as network topology monitoring, bottleneck detection and rerouting, service level agreement reporting, and accounting. Construction procedures and graphical representation of the edge-to-edge traffic aggregates are explained in detail. We also extend our efforts to implement a DiffServ domain monitoring system that monitors a set of DiffServ-enabled routers and traffic aggregates between every edge router pair.

The organization of the paper is as follows. Section 2 is devoted to presenting the current standardization effort of DiffServ and Section 3 shows how to manage a DiffServ router with the DiffServ MIB in the SNMP framework. Section 4 describes our construction procedures including graphical representations and aggregation rules in detail. We also describe a DiffServ domain monitoring system that monitors DiffServ-enabled routers and edge-to-edge traffic aggregates in Section 5. In Section 6, various related work is reviewed and compared with our approach. Finally, we conclude and discuss possible future work in Section 7.

2. BASIC DIFFSERV ARCHITECTURE

The IETF DiffServ working group has been established in 1998 and has defined the basic architecture of DiffServ [3]. In this section, we summarize current standards and on-going research efforts of the working group.

2.1. PHB and DSCP

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 [6–9]. They are default, class-selector, Assured Forwarding (AF), and Expedited Forwarding (EF) PHBs. Table I 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.

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 condi-

Table I. Standard PHB and Reserved DiffServ Code Points[a]

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

[a] Bits in the first column: first three bits in DSCP; bits in the first row: last three bits in DSCP.

tion) 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.

2.2. Traffic Conditioning Block

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 1 illustrates four components of a traffic conditioning block [3].

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.

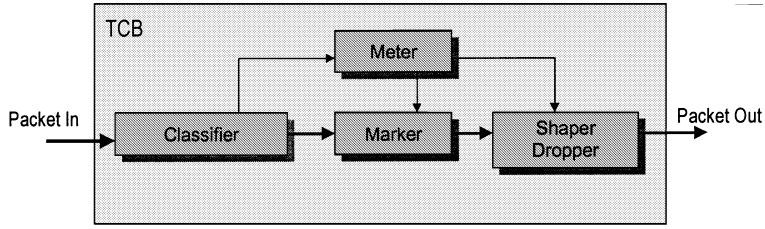

Fig. 1. Basic traffic conditioning block of DiffServ.

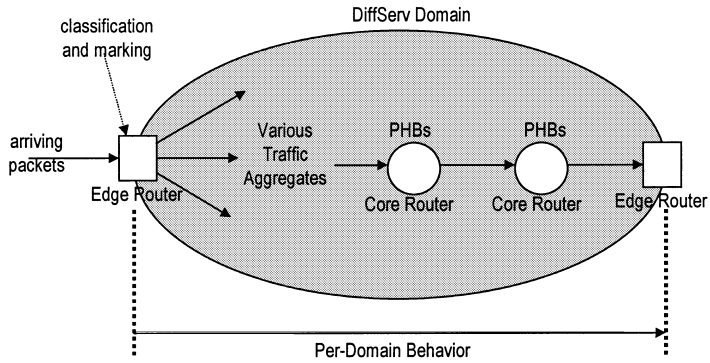

Fig. 2. PHB and PDB relationship in a DiffServ domain.

2.3. Per-Domain Behavior

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 2 illustrates the relationship of PHB and PDB in a DiffServ domain.

Using the classification and marking performed at edge routers, arriving packets are mapped to a set of traffic aggregates 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 with 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 for-

warding in a DiffServ domain, an extended approach to describe the behaviors is needed.

We think that 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. In the following sections, we explain how to manage DiffServ routers and to monitor edge-to-edge DiffServ traffic aggregates in a DiffServ domain.

3. MANAGING A DIFFSERV ROUTER WITH DIFFSERV MIB

The management of a DiffServ network starts from managing a DiffServ router in its domain. The SNMP framework is chosen for the DiffServ management platform. The SNMP is a simple, cost-effective, and *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 DiffServ working group.

3.1. Conceptual Model of a DiffServ Router

A DiffServ router is a fundamental DiffServ-enabled network node. The conceptual model and requirements of the DiffServ routers are discussed by Bernet *et al.* [12, 13]. A DiffServ router is considered to have a routing module, a set of TCBs, a queuing module, and a configuration and monitoring module that are organized as in Fig. 3.

DiffServ-related modules are separated from the routing module to simplify the addition of the DiffServ capability to the existing router. Traffic conditioning

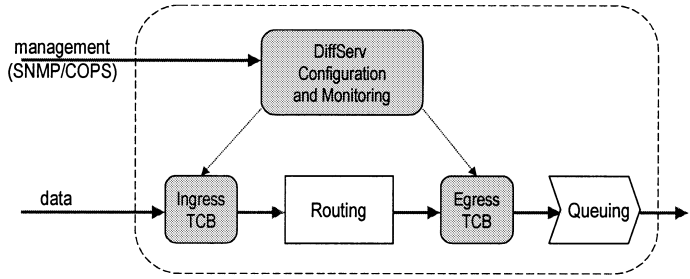

Fig. 3. Conceptual model of a DiffServ router.

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 [14, 15]. The management module configures TCB parameters and monitors the performance of each TCB. Therefore, managing a DiffServ router means managing the TCB parameters in every ingress/egress interface.

3.2. IETF DiffServ MIB

The IETF DiffServ working group currently suggests an SNMP Management Information Base (MIB) for DiffServ architecture. The MIB is designed according to the DiffServ implementation conceptual model [13] for managing DiffServ routers in the SNMP framework. The initial draft was proposed in July 1999 and several extensions have been added. Detailed definitions are still being elaborated and extended in the working group. Table II summarizes the primary object tables defined in the DiffServ MIB version 3 [5].

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 II are subordinate tables that belong to a 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 to

Table II. 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[a]	absolute drop action
Queue	AlgDrop	algorithmic dropper parameters
	RandomDrop	random dropper parameters
	Queue	queuing parameters
	Scheduler	scheduling parameters

[a] AbsoluteDrop is an object name, not a table name.

be 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. The 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 to describe 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 4 shows the RowPointer linked relationship of tables in the DiffServ MIB.

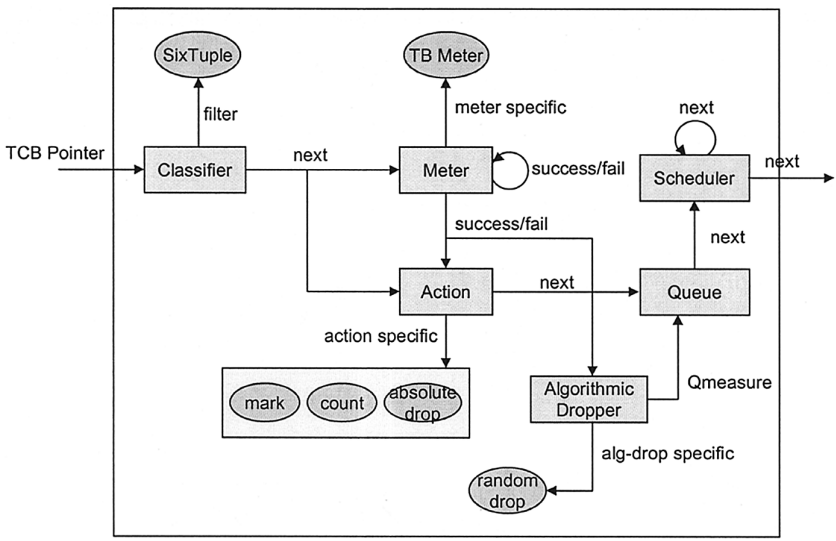

Fig. 4. DiffServ MIB table RowPointer linked relationship in a TCB.

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 Fig. 3, 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.

4. MONITORING EDGE-TO-EDGE TRAFFIC AGGREGATES

After edge routers in DiffServ networks classify incoming packets to a set of traffic aggregates by marking appropriate DSCP values in the packet header, DiffServ routers perform traffic conditioning operations on traffic aggregates distinguished by DSCP values, not on the individual traffic flows distinguished by 5-tuple header information. A traffic aggregate is defined as a collection of packets with a codepoint that maps to the same PHB, usually in a DiffServ domain or a related subset. Different traffic aggregates receive different treatment as they pass through the DiffServ domain.

Every network service provided from a DiffServ network can be represented as a traffic aggregate from a set of source nodes to a set of destination nodes. For example, let us consider a VOD service from a server to a set of customer clients. Edge routers handling the VOD server and clients mark the packets with a certain DSCP value. The marked packets are delivered by DiffServ core routers, which follow predefined PHBs. At this stage, the DiffServ network is carrying traffic aggregates for the VOD service from a VOD server to a set of customer clients.

Managing traffic aggregates in a single DiffServ router can be achieved by managing the DiffServ MIB, as shown in the previous section. However, man-

aging traffic aggregates in a DiffServ domain requires many more additional functions. Since, as in the previous example, every network service spans over multiple DiffServ routers in a DiffServ domain, properties and information from all the routers participating in the network service should be combined and incorporated in a systematic way.

From this consideration, we suggest that edge-to-edge traffic aggregates should be constructed and monitored for providing sophisticated management functions. We define an edge-to-edge traffic aggregate as a traffic aggregate flowing from one set of edge router(s) to another set of edge router(s) in a DiffServ domain. Information obtained from edge-to-edge DiffServ traffic aggregates 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.

When managing DiffServ networks, it is beneficial to possess information on edge-to-edge traffic aggregates in a DiffServ domain. First, edge-to-edge traffic aggregates 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 traffic aggregates 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 traffic aggregates. Secondly, routing decisions can be combined with traffic conditioning decisions. Without edge-to-edge traffic aggregates, 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 traffic aggregates 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 traffic aggregates. Dynamically changing behaviors can be classified in the traffic aggregates, and administrators can easily understand and manipulate network traffic by managing traffic aggregates, not by managing individual routers. This will also help to configure, control, and monitor PDBs in a DiffServ domain.

In this section, we suggest a method to create edge-to-edge DiffServ traffic aggregates by combining routing information from MIB II and performance parameters from DiffServ MIB. The edge-to-edge DiffServ traffic aggregate information can be used as a basic component for providing sophisticated high-level management functions.

4.1. Construction of Edge-to-Edge Traffic Aggregates

An edge-to-edge traffic aggregate 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 values in each DiffServ router. Constructing edge-to-edge DiffServ traffic aggregates thus consists of two phases, as in Fig. 5. 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 and DiffServ MIB are used to construct the edge-to-edge DiffServ traffic aggregates.

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

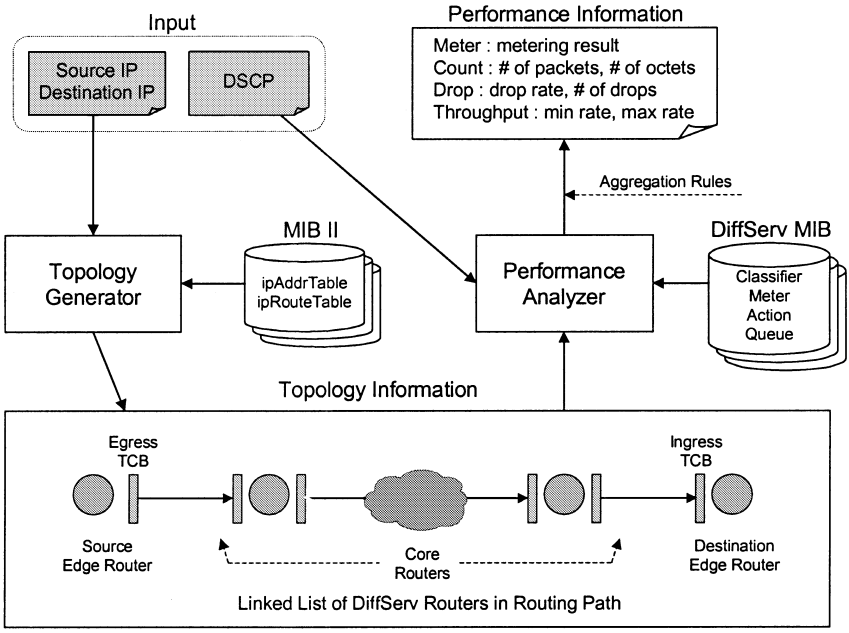

Fig. 5. Construction process of edge-to-edge DiffServ traffic aggregates.

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 on edge-to-edge traffic aggregates 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 traffic aggregates 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 traffic aggregate.

For example, the overall packet drop rate of an edge-to-edge DiffServ traffic aggregate can be calculated by accumulating drop rates of the DSCP-marked packets in every router. Further, the guaranteed minimum packet transmission rate of an edge-to-edge DiffServ traffic aggregate can be calculated by finding the minimum value among minimum rates of all routers in the routing path.

One important consideration in calculating edge-to-edge performance information is that the performance parameters contained in the DiffServ MIB in each router do not distinguish packets with a different IP source/destination pair. Defined by the DiffServ concept, every core router forwarding packets from the source node to the destination node merely looks up the DSCP value in the header of each packet. Thus, performance parameters from the DiffServ MIB are for aggregated traffic with a given DSCP value, not for a specific traffic aggregate from a given source to a given destination, which we want to analyze. The traffic aggregate we wish to distinguish is mixed with other traffic aggregates with the same DSCP value but with different source/destination pairs. From this observation, we formulate rules to follow when aggregating performance parameters from DiffServ MIB to extract edge-to-edge traffic aggregates of concern.

4.2. Graphical Representation and Aggregation Rules

To represent edge-to-edge DiffServ traffic aggregates more efficiently, we devised a graphical notation of topology. We show basic graphical notations of DiffServ traffic aggregates to understand the topological information of each traffic aggregate. Figure 6 illustrates four different kinds of DiffServ node representations, composed of vertex and directed edges.

A direct node receives the traffic aggregate from one direction and forwards it to only one direction. A split node receives the traffic aggregates from one

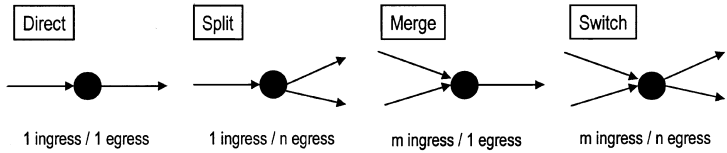

Fig. 6. Graphical representation of DiffServ nodes.

direction but forwards them to two or more directions. A merge node receives the traffic aggregate 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.

The hop-by-hop aggregation rules are used when the performance analyzer construct edge-to-edge performance information following the routing path. Locally observed performance parameters from DiffServ MIB are aggregated differently according to the kind of node in the routing path. Table III summarizes the different aggregation rules for each parameter and node combination. Three performance parameters, throughput, delay, and number of drops, are explained as examples.

There are four different operations for building up aggregation rules. One aggregation rule consists of one or several 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 traffic aggregate 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

Table III. Aggregation Rules for Different Parameter/Node Combination

	Direct	Split	Merge	Switch

Throughput	`find_min`	`find_next_hop`
		`find_min`
			`calculate_portion`
			`find_min`
				`find_next_hop`
				`calculate_portion`
				`find_min`

Delay	`add`	`find_next_hop` `add`	`add`	`find_next_hop` `add`

Number of drops	`add`	`find_next_hop`
		`add`
			`calculate_portion`
			`add`
				`find_next_hop`
				`calculate_portion`
				`add`

delay of a link and number of dropped packets can be calculated by summing up all the values on the routing path. For the nodes with multiple egress links, such as split and switch nodes, a `find_next_hop` operation must be performed before other operations to specify one egress link that the edge-to-edge traffic aggregate of concern actually uses for the next routing path. For the nodes with multiple ingress links, such as merge and switch nodes, a `calculate_portion` 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 traffic aggregate 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.

$$\text{portion} = \frac{\text{current throughput of traffic aggregate in the ingress link}}{\text{throughput of the next hop egress link}}$$

The `calculate_portion` operation is important for extracting the edge-to-edge traffic aggregate from mixed traffic aggregates in a link. If there are multiple ingress links and the DiffServ node does not distinguish different edge-to-edge traffic aggregates because of the same DSCP value, every egress link might have a mix of different edge-to-edge traffic aggregates. However, since we know the current amount of throughput of the edge-to-edge traffic aggregate 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 the 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 traffic aggregate of concern is 10 Mbps in the ingress link, while the throughput of the next hop egress link is 20 Mbps, we assume that half of the egress link traffic comes from other edge-to-edge traffic aggregates. In this situation, the result of the `calculate_portion` 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 traffic aggregate of concern. A certain kind of performance parameters, such as delay parameter, does not use the `calculate_portion` operation because of its inherent characteristics.

Figure 7 is a graphical representation of an example of DiffServ traffic aggregates snapshot. In this snapshot, three source edge routers (A, B, and C) and two destination edge routers (D and E) exist. Six core routers (labeled 1 through 6) are interconnected with each other by directed edges representing IP routing paths. According to our graphical representation, router 1 and 4 are direct nodes, router 5 is a split node, router 3 and 6 are merge nodes, and finally router 2 is a switch node.

The edge-to-edge traffic aggregate from edge router A to edge router D is

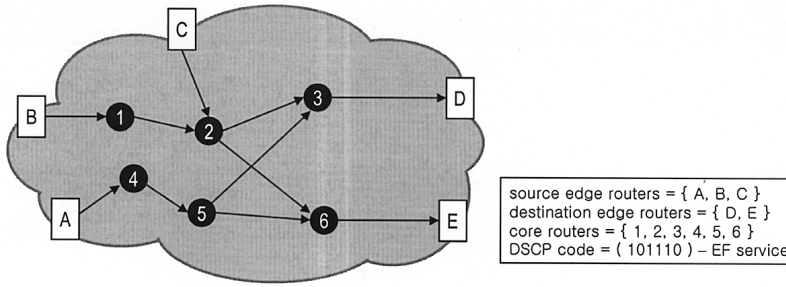

Fig. 7. Example of edge-to-edge DiffServ traffic aggregates.

constructed as follows. According to IP routing tables, the topology generator extracts the routing path from router A to router D as A – 4 – 5 – 3 – D. Since router 4 is a direct router, performance parameters are simply aggregated at the point. Next, router 5, the traffic aggregate is separately forwarded to core routers 3 and 6, and the performance parameters from router A to router 4 is divided to reflect the division. Router 3, a merge router, combines traffic aggregates from 2 and 5, and the performance parameters from router 3 to router D is divided to reflect the aggregation.

5. DEVELOPING A DIFFSERV DOMAIN MONITORING SYSTEM

In this section, we present a detailed design and implementation of a DiffServ domain monitoring system based on the SNMP framework.

5.1. Design Architecture

The architecture consists of three distinct parts, as depicted in Fig. 8. The three-tier architecture includes a network management system (NMS) client running as a monitoring console, an NMS server containing a DiffServ domain monitoring system, and network elements performing DiffServ routing and SNMP agent functionality.

The NMS server is a central server used to monitor a set of DiffServ routers and to provide user interfaces to a set of monitoring consoles. The DiffServ domain monitoring system performs three monitoring modules—topology generator, performance analyzer, and edge-to-edge traffic aggregates monitor. A monitoring database is needed for storing and retrieving the combined and analyzed data from MIB II and DiffServ MIB. At the bottom of the DiffServ domain monitoring system, an SNMP manager communicates with a set of SNMP agents running in different DiffServ routers within a DiffServ domain.

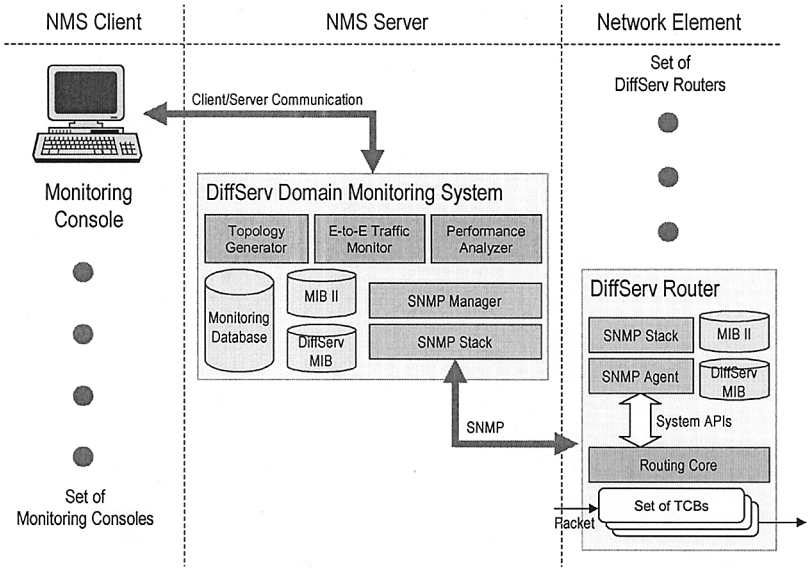

Fig. 8. Design architecture of the DiffServ domain monitoring system.

The 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. Different methods for retrieving and setting DiffServ parameters in the routing core may be employed for given different implementation architectures.

Within a DiffServ domain, numerous DiffServ routers and DiffServ monitoring clients interwork with each other. This three-tier architecture offers distinct advantages in such network management environments. One centralized DiffServ domain monitoring system oversees a set of DiffServ routers while providing user interfaces to a set of monitoring clients at the same time. However, by separating the monitoring user interfaces from the monitoring system itself, the DiffServ domain monitoring system is able to concentrate on monitoring functions and thus improve monitoring performance. To address the scalability problem, the three-tier architecture can be extended to support distributed management functionality with multiple DiffServ monitoring systems located in the middle.

5.2. Implementation

In order to realize our system design, 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 monitoring system, 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 domain monitoring console has also been developed for delivering various monitoring services to human administrators.

5.2.1. Linux DiffServ Agent

Linux, a shareware operating system, supports QoS features in its networking kernel from the kernel version 2.1.90 [17]. 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 [18]. 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 [19] 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.

However, the current Linux DiffServ implementation does not show sufficient management functionality. No management architecture exists and every script setup must be manually configured and modified in local machines. Further, metering and monitoring functions of DiffServ are not fully supported. Our work has focused on this lack of management functionality.

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 based 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 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 achieved by using 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.

The agent has been implemented by using a UCD SNMP agent extension package [21]. UCD SNMP 4.1.2 provides the agent development environment. The DiffServ agent uses a traffic control program (tc) or a NetLink socket directly for accessing DiffServ parameters in kernel space and manipulates the values of MIB II and DiffServ MIB.

5.2.2. DiffServ Domain Monitoring System and Monitoring Console

The central DiffServ domain monitoring system is implemented in the Linux platform with C language. UCD-SNMP management APIs are used for handling SNMP manager operations. 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 [22]. The central monitoring system sends SET and GET requests to DiffServ agents in DiffServ routers under its control in order to access 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 monitoring consoles. Topology generator, performance analyzer, and edge-to-edge traffic aggregates monitor are three distinct functional modules implemented in the central monitoring system.

The monitoring 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 OS platforms. If Web integration is highly recommended for ubiquitous access, the management console can be easily adjusted to Java applets running on Web browsers. However, standalone Java applications show a quicker and more stable execution performance than Java applets running on Web browsers. In Fig. 9, two example screenshots of the Java-based DiffServ domain monitoring console are displayed.

The left screenshot shows a general desktop display of the DiffServ domain monitoring console. On the left of the screen, a set of routers and a set of DiffServ traffic aggregates are managed in tree-style directories. Detailed information and operation results are shown on the right of the screen, which shows an edge-to-edge DiffServ traffic aggregate in the topology viewer. The viewer depicts router connectivity and the routing path of a specified traffic aggregate from a given edge router pair. Various topology and performance parameters can be retrieved from the viewer screen. Thus, the status of DiffServ services supported by the given traffic aggregate can be easily understood and managed.

Communication between the monitoring console and the central monitoring system is achieved using a proprietary application protocol that we developed over TCP sockets. Monitoring operations and response data are transferred by the protocol. The protocol also supports multiple concurrent monitoring consoles and password security for providing different console views to different human administrators.

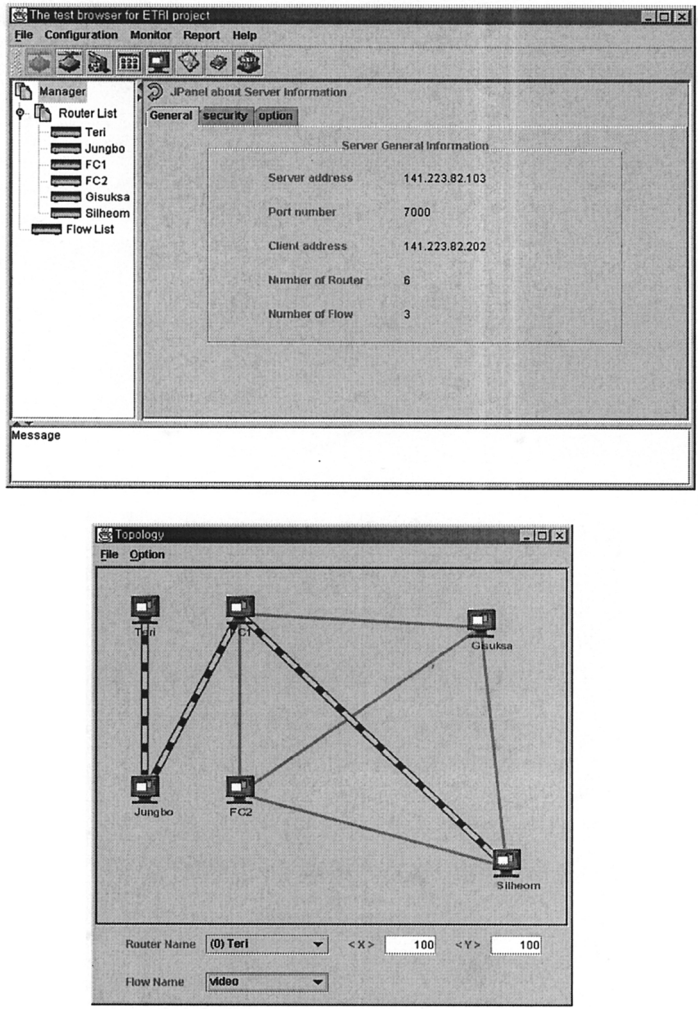

Fig. 9. Screenshots of the DiffServ domain monitoring console.

6. RELATED WORK

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

IETF's real-time flow measurement (RTFM) architecture [23, 24] and Cisco's NetFlow [25] 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 [26] and its extension RMON2 [27] 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.

Jiang *et al.* [28] 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 in showing how to construct network-wide flow information in detail.

Feldmann *et al.* [29] 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 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 system is to show edge-to-edge performance information for each DiffServ traffic aggregate. Both systems are able determine which network link is overloaded, but; only our system is able to determine the loaded traffic comes from and where it sinks to because our system has performance information of every edge-to-edge traffic aggregate. Further, the NetScope system has been applied to general IP networks while our system has been applied to QoS-enabled DiffServ networks. Our system is able to differentiate various traffic classes and monitor different QoS parameters.

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

7. CONCLUSIONS

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.

We have proposed a generalized mechanism to monitor DiffServ networks using the SNMP framework. First, we have overviewed the general architecture of DiffServ and then summarized ongoing work to define MIB for managing DiffServ-enabled network nodes in the IETF working group. The DiffServ MIB has been analyzed in detail to show how to manage a single DiffServ router. We have also proposed a method to construct and maintain edge-to-edge DiffServ traffic aggregates by combining information from MIB II and DiffServ MIB. For providing network-wide status of a DiffServ domain, we have devised a graphical representation and aggregation rules for edge-to-edge traffic aggregates for easier maintenance. We also proposed a DiffServ domain monitoring system with a flexible three-tier architecture using the SNMP framework. To validate our design, we have developed a DiffServ agent system working in a Linux platform and a DiffServ domain monitoring system composed of a Java-based monitoring console and a central monitoring system that monitors a set of DiffServ agent systems. We believe the proposed system architecture can be easily deployed in DiffServ networks.

To improve the proposed DiffServ monitoring system, we are currently concentrating on the following research areas:

- 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, 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. The PDB description is a possible candidate to represent edge-to-edge traffic aggregates.
- The scalability of the proposed system should be improved. The current monitoring method needs repeated polling to every router in a DiffServ domain. This might not be appropriate, especially in big ISP backbones. To address the scalability problem, the three-tier architecture can be extended to support distributed functionality with multiple DiffServ

domain monitoring systems located in the middle. Interactions among the systems for sharing monitoring information should be clearly stated. Further, instead of a SNMP manager polling the routing table, the agent can initiate sending routing change notification to managers by using the SNMP trap method. However, since Internet backbone networks of a single administrative domain usually have less than 100 routers, the proposed system architecture can cope with most DiffServ domains.

- A performance evaluation of the monitoring system we developed is being considered. Because general DiffServ routers handle a huge amount of high-speed traffic, the DiffServ agent must not affect the routing performance of the DiffServ routers. A DiffServ domain monitoring system must be implemented in such a way as to minimize performance degradation factors.
- Recently, Multiprotocol Label Switching (MPLS) [30–32] is being considered as a practical switching technology for providing virtual circuits in Internet backbones. 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. We are considering applying our monitoring methods in MPLS domains and comparing the results with those of DiffServ domain.

ACKNOWLEDGMENTS

The work described in this paper was supported in by the Ministry of Information and Communication of Korea and by the Ministry of Education of Korea for its financial support toward the Electrical and Computer Engineering Division at POSTECH through its BK21 program.

REFERENCES

1. L. Mathy, C. Edwards, and D. Hutchison, The Internet: A global telecommunications solution?, *IEEE Network*, pp. 46–57, July/August 2000.
2. R. Braden, D. Clark, and S. Shenker, Integrated services in the Internet architecture: An overview, IETF RFC 1633, June 1994.
3. S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, An architecture for differentiated services, IETF RFC 2475, December 1998.
4. R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, ReSerVation protocol (RSVP) Version 1 functional specification, IETF RFC 2205, September 1997.
5. 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.
6. J. Heinanen, Use of IPv4 TOS Octet to support differential services, IETF Internet-Draft, *draft-heinanen-diff-tos-octet-01.txt*, November 1997.
7. 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.

8. J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski, Assured forwarding PHB group, IETF RFC 2597, June 1999.
9. V. Jacobson, K. Nichols, and K. Poduri, An expedited forwarding PHB, IETF RFC 2598, June 1999.
10. K. Nichols and B. Carpenter, Definition of differentiated services per domain behaviors and rules for their specification, IETF Internet Draft, *draft-ietf-diffserv-pdb-def-00.txt*, June 2000.
11. V. Jacobson, K. Nichols, and K. Poduri, The 'Virtual Wire' Per-Domain Behavior, IETF Internet Draft, *draft-ietf-diffserv-pdb-vw-00.txt*, July 2000.
12. Y. Bernet, D. Durham, and F. Reichmeyer, Requirements of Diff-serv boundary routers, IETF Internet-Draft, *draft-bernet-diffedge-01.txt*, November 1998.
13. 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.
14. 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.
15. R. Yavatkar, K. McCloghrie, S. Herzog, F. Reichmeyer, D. Durham, K. Chan, and S. Gai, COPS usage for differentiated services, IETF Internet-Draft, *draft-ietf-rap-cops-ds-00.txt*, August 1998.
16. W. Stalling, *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2*, Third Edition, Addison-Wesley, 1999.
17. S. Radhakrishnan, Linux—Advanced networking overview—Version 1, Technical paper, Department of Electrical Engineering and Computer Science, University of Kansas, August 22, 1999.
18. W. Almesberger, J. H. Salim, and A. Kuznetsov, Differentiated services on Linux, IETF Internet-Draft, *draft-almesberger-wajhak-diffserv-linux-01.txt*, June 1999.
19. W. Almesberger, Differentiated services on Linux, Internet Web site, *http://lrcwww.epfl.ch/linux-diffserv/*.
20. G. Dhandapani and A. Sundaresan, Netlink sockets—Overview, Technical paper, Department of Electrical Engineering and Computer Science, University of Kansas, September 13, 1999.
21. UCD-SNMP Homepage, *http://ucd-snmp.ucdavis.edu/*.
22. PostgreSQL Homepage, *http://www.postgresql.org/*.
23. N. Brownlee, C. Mills, G. Ruth, Traffic flow measurement: Architecture, IETF RFC 2063, January 1997.
24. N. Brownlee, Traffic flow measurement: Experiences with NeTraMet, IETF RFC 2123, March 1997.
25. Cisco NetFlow white paper, *http://www.cisco.com/warp/public/cc/pd/iosw/ioft/neflct/tech/napps wp.htm*.
26. S. Waldbusser, Remote network monitoring management information base, IETF RFC 2819, May 2000.
27. S. Waldbusser, Remote network monitoring management information base version 2 using SMIv2, IETF RFC 2021, January 1997.
28. Y. Jiang, C. Tham, and C. Ko, Challenges and approaches in providing QoS monitoring, *International Journal of Network Management*, October 2000, pp. 322–334.
29. A. Feldmann, A. Greenberg, C. Lund, N. Reingold, and J. Rexford, NetScope: Traffic engineering for IP networks, *IEEE Network*, Vol. 14, No. 2, pp. 11–19, March/April 2000.
30. X. Xiao, A. Hanna, B. Bailey, and L. M. Ni, Traffic engineering with MPLS in the Internet, *IEEE Network*, Vol. 14, No. 2, pp. 28–33, March/April 2000.
31. 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*, pp.49–53, December 1999.
32. G. Swallow, MPLS advantages for traffic engineering, *IEEE Communications Magazine*, Vol. 37, No. 12, pp.54–57, December 1999.

Jae-Young Kim received B.S. and M.S. degrees from the Pohang University of Science and Technology (POSTECH) in 1994 and 1996, respectively. From 1996 to 1997 he was a researcher in Computing Center of the same University and worked in intelligent campus and digital library projects. Since 1998 he has been pursuing his Ph.D. degree in the Department of Computer Science and Engineering at POSTECH, Pohang, Korea. His research interests include differentiated service networks, 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. Since joining POSTECH in 1995, 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 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.

Tae-Sang Choi received M.S. and Ph.D. degrees from the University of Missouri-Kansas City in 1991 and 1995, respectively. Since 1996, he has been a senior research staff in Electronics and Telecommunications Research Institute (ETRI) in Korea and has worked in high-quality real-time video-on-demand and Internet QoS management and traffic engineering system development projects. His area of research interests includes high quality multimedia systems, Internet QoS management, Internet traffic engineering, and network and service management.