Layer-4 Service Differentiation and Resource Isolation*

Haining Wang and Kang G. Shin

Real-Time Computing Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, MI 48109-2122
{hxw, kgshin}@eecs.umich.edu

Abstract

While the Differentiated Services (DiffServ) infrastructure is scalable and robust in providing network Quality of Service (QoS), there are serious drawbacks with the services provided by DiffServ: (1) the services are coarse-grained and one-way only; (2) no service differentiation and resource isolation are provided to meta-data packets such as TCP SYN and ACK packets. Moreover, the coarse-grained service differentiation and the lack of resource isolation at IP routers exposes its vulnerability to Distributed Denial of Service (DDoS) attacks [10]. Based on the concept of layer-4 service differentiation and resource isolation, where the transport-layer information is inferred from the IP headers and used for packet classification and resource management, we present a scalable fine-grained DiffServ (sf-DiffServ) architecture that provides fine-grained service differentiation and resource isolation among thinner Behavior Aggregates (BAs). The sf-DiffServ architecture consists of a fine-grained QoS classifier and an adaptive weight-based resource manager at IP routers. A two-stage packet classification mechanism is devised to decouple the fine-grained QoS lookup from the routing lookup at core routers. Due to its scalable QoS support for TCP control segments, sf-DiffServ supports bi-directional differentiated services for TCP sessions. Most importantly, the fine-grained resource isolation provided inside the sf-DiffServ is a powerful built-in protection mechanism to counter DDoS attacks, reducing the vulnerability of Internet to DDoS attacks.

1 Introduction

As the Internet evolves into a ubiquitous communication infrastructure, more sophisticated services than the traditional best-effort service have become necessary to meet the Quality-of-Service (QoS) requirements of various real-time applications. To support network QoS, the Differentiated Services (DiffServ) infrastructure [5] has been proposed as a promising solution due mainly to its scalability and robustness. Based on the DS field in the IP header, IP flows are classified into different Behavior Aggregates (BAs). Services are provided for aggregates, not for individual flows, and defined by a small set of Per-Hop Behaviors (PHBs), which are the forwarding behaviors applied to different aggregates at IP routers. The current best-effort service at IP routers is replaced by three different services: best-effort service and two more advanced services, i.e., premium and assured services. Corresponding to the three different services provided by DiffServ, three types of PHBs are specified: *Expedited Forwarding* (EF), *Assured Forwarding* (AF), and *Best-Effort* (BE). In the DiffServ architecture, EF is to support premium service for stringent (hard) real-time applications that require bounded end-to-end delay and jitter, and AF is to support assured service for soft real-time applications.

The current DiffServ architecture only provides one-way differentiated service for each session, which works well only if UDP is used for transporting application data [35]. However, besides HTTP, a large fraction of application-layer protocols like ftp and telnet, are based on TCP for reliable data transport service. The latest Internet traffic measurements have shown that 90–95% of the Internet traffic belongs to TCP [8]. The reliability of TCP is achieved by its three-way handshake, acknowledgment (ACK) and timeout mechanisms. The TCP congestion control mechanism [12, 34] makes the data transmission of a TCP connection ACK-clocked. The success of meeting the application's deadline depends not only upon the service guaranteed for TCP segments in their forwarding path, but also upon the service received by the returning TCP ACKs in the backward path. Unfortunately, the current DiffServ architecture provides no service differentiation and resource isolation to meta-data packets such as TCP SYNs and ACKs. By default, the current DiffServ treats these TCP control segments from different user classes as BE traffic. Recent experiments [22, 35] have shown that the current DiffServ can not meet the required service assurance for TCP-based applications that need bi-directional service differentiation.

*The work reported in this paper was supported in part by Samsung Electronics, Inc. and the US Office of Naval Research under Grant N00014-99-1-0465.

Proceedings of the Eighth IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'02)
0-7695-1739-0/02 $17.00 © 2002 IEEE

Moreover, in the current DiffServ architecture, the QoS classification at core routers solely depends upon the DS field in the IP header,[1] yielding only coarse-grained service differentiation and resource isolation. No further service differentiation and resource isolation are provided among different transport-layer protocols within a BA. UDP and TCP are two dominant transport-layer protocols in the current Internet, but their services and traffic behaviors are quite different. It is beneficial to divide a single BA into a UDP behavior aggregate and a TCP behavior aggregate. Besides meeting the requirement of the bi-directional service differentiation to TCP sessions in the context of DiffServ, there are three reasons for differentiating TCP control segments — SYNs, FINs, ACKs and RSTs — from data segments even in the best-effort service model: (1) usually TCP control segments have much smaller packet size than that of data segments; (2) the loss of a TCP control segment incurs more serious performance degradation than the loss of a data segment; (3) DDoS attack tools usually utilize TCP control segments for generation of DoS attacks, such as TCP SYN and ACK flooding attacks. In other words, the coarse-grained service differentiation and the lack of resource isolation on meta-data packets not only degrade the assured service of TCP sessions but also expose the vulnerability of Internet to DDoS attacks [10].

In this paper, we propose a scalable fine-grained DiffServ (sf-DiffServ) architecture to provide fine-grained service differentiation and resource isolation among thinner aggregates without compromising scalability. The basic concept employed is layer-4 service differentiation and resource isolation, in which the transport-layer information is inferred from the IP headers and used for packet classification and resource management at IP routers. To support layer-4 service differentiation and resource isolation, we present a fine-grained QoS classifier and an adaptive weight-based resource manager, with which the scalable fine-grained DiffServ infrastructure is built. The fine-grained QoS classifier divides each BA into thinner aggregates, and the adaptive weight-based resource manager provides service differentiation and resource isolation among these thinner aggregates. At core routers, we employ a two-stage packet classification mechanism to decouple the routing lookup from QoS lookup. The first stage performs the routing lookup at the input port, and the second stage performs the fine-grained QoS lookup at the output port after the packet is routed through the switching fabric. The service differentiation between different TCP control flows is relative, instead of quantitative or qualitative. It guarantees that the TCP control segments for high-tiered TCP sessions receive better service (i.e., lower delay and lower loss rate) than those for low-tiered sessions. No bi-directional resource reservation is needed.

The performance of the sf-DiffServ architecture is evaluated by simulation. The simulation results show that the sf-DiffServ provides better end-to-end QoS to applications: the response time of a premium request is guaranteed to be shorter than that of basic requests with the same round-trip time (RTT); a high-tiered TCP session has lower ACK loss rate and higher effective throughput than a low-tiered one. Moreover, the resource isolation provided by the sf-DiffServ significantly throttles the flooding traffic received by the victim server, which can be utilized as a built-in protection mechanism of routers to counter DDoS attacks.

The remainder of the paper is organized as follows. Section 2 presents the motivation of this work and the related work. Section 3 describes a fine-grained QoS classifier and an adaptive weight-based resource manager, the key components of the sf-DiffServ architecture. Section 4 evaluates the performance of the sf-DiffServ architecture. Finally, the paper concludes with Section 5.

2 Motivation and Related Work

Our work was initially motivated by the desire of providing preferential treatments to TCP ACKs and achieving bi-directional differentiated service to a TCP session. One simple way to achieve this is that end-hosts mark each ACK as a premium, assured or best-effort packet, corresponding to the class of the data packet being acknowledged, but no enhanced mechanisms implemented at IP routers to distinguish ACKs from data segments.

The validity of this simple marking scheme has been confirmed by the authors of [22], where the effect of ACK marking on TCP throughput is thoroughly studied. They found that copying the marking of a data segment into its ACK is a practical and optimal strategy if resource provisioning is done properly for EF and AF aggregates. At the same time, they pointed out the necessity of the resource provisioning for EF and AF ACK flows. However, the proper resource provisioning requires the accurate traffic profile of the ACK flows. Unfortunately, it is extremely difficult to characterize the traffic profile of ACK flows, such as peak rate and burst size, because the traffic characterization of an ACK flow depends on the following factors that are not known until the TCP connection has been established: (1) the actual arrival rate of TCP data segments in the forward path; (2) the processing time required for the receiver to generate an ACK for a newly-received data segment — delayed acknowledgment or not; (3) the Maximum Segment Size (MSS) in the forward path, which determines the number of TCP data segments and, implicitly, the number of TCP ACKs; and (4) the loss rate of data segments in the forward path.

Without proper resource provisioning and traffic conditioning for ACK aggregates, the ACKs and data segments that share the same queue could interfere with each other. For EF or AF aggregates, adding unconditioned ACK flows could cause the violation of their traffic profiles at routers. In other words, the conformant EF or AF data flows may suffer from packet losses, longer delay and delay jitter, as a result of adding unconditioned ACK flows. Especially, under

[1] Multi-field packet classification is limited to edge routers in the DiffServ architecture.

distributed ACK flooding attacks, the performance of conformant EF or AF flows will seriously degrade. On the other hand, an ACK flow could also be interfered with by non-conforming EF or AF data flows, resulting in unnecessary ACK losses by the traffic shaper at boundary routers. Our simulation results shown in Section 4.2 demonstrate that the simple ACK marking scheme provides insufficient service differentiation and isolation to ACK flows when network resources are under-provisioned. Also, sharing the same queue with large data segments leads to ACK compression [38].

Furthermore, there are two serious drawbacks with this simple marking scheme: (1) the best-effort TCP traffic, which will continue to be the dominant load on the Internet, will not receive any performance improvement with the simple marking scheme; (2) DDoS attacks on the Internet make the simple marking scheme much less attractive, since it exposes more vulnerability to various TCP flooding attacks and upgrading the service level of TCP control segments will make the problem even worse. The simulation results shown in Section 4.3 confirm this claim.

To improve the TCP performance in the context of network asymmetry, the *acks-first* scheduling scheme has been proposed [4], giving TCP ACKs priority over TCP data packets. So, the router always forwards ACKs before data segments. However, this *acks-first* scheme could cause starvation of data packets and violation of traffic profiles, especially under ACK flooding attacks. Also, no ACK identification scheme at routers was provided in [4]. Therefore, to achieve better network QoS and counter DDoS attacks, we need to differentiate the TCP control segments from data segments at IP routers.

Resource management is essential to real-time applications. Meeting timing constraints with high resource utilization is the a key goal of resource management. Numerous approaches have been proposed and implemented to achieve this goal [1, 2, 3, 9, 19, 25, 30, 36, 37], in which adaptation and hierarchy are the two key features. Also, resource isolation has been utilized as a powerful mechanism to counter DoS attacks [3, 17, 27]. However, most of these approaches are designed for applications, end-hosts and edge routers, without paying attention to the resource management at core routers due mainly to the scalability problem. In this paper, based on the layer-4 classification, we build a flexible and scalable resource manager at core routers without requiring a resource signaling protocol like RSVP [39] in the DiffServ architecture.

3 Scalable Fine-grained DiffServ Architecture

To provide layer-4 service differentiation and isolation, we propose a fine-grained QoS classifier and an adaptive weight-based resource manager, both of which are essential to the sf-DiffServ architecture. The granularity of the classifier is still based on aggregates, not individual flows, and the resource manager is stateless, thus preserving the scalability and robustness of the original DiffServ infrastructure. Moreover, at core routers the QoS lookup is decoupled from the routing lookup by employing a two-stage packet classification mechanism. The set of QoS filtering rules is small, and has the same/similar size at both edge and core routers. In contrast to routing lookup, QoS lookup is independent of network size and does not cause any scalability problem in packet classification. The sf-DiffServ architecture is detailed in the remainder of this section.

3.1 Two-stage packet classification

For service differentiation, layer-4 switching [14, 28] has been proposed, in which routing decisions are made based not only on the destination address but also on the header fields at the transport or higher layer. Routing and QoS lookups are integrated into a single framework to fulfill layer-4 switching, and therefore, the forwarding database of a router consists of a large number of filters to be applied on multiple header fields. The deployment of a large-scale packet filtering mechanism [11, 13, 28] makes it feasible to implement layer-4 or 5 switching at edge routers or at the front-end of server farms. However, layer-4 switching has primarily been used for load balancing by connection routers in server farms. It is very difficult to implement layer-4 switching at core routers due mainly to security and scalability difficulties. Even with fast and scalable packet classification, the problems with layer-4 switching at core routers are: (1) addition of higher-layer information — such as port numbers — and more routing entries enlarges the routing table at core routers, causing the routing lookup to require much more memory and time; (2) when IP payload is encrypted, higher-layer headers become inaccessible.

To support layer-4 service differentiation and resource isolation, the fine-grained QoS classification has to work well at both edge and core routers. Therefore, we decouple the fine-grained QoS lookup from the routing lookup at core routers by employing a two-stage packet classification mechanism. In addition to overcoming the scalability problem at core routers, there are several other reasons for this decoupling.

- Routing decisions must be made at the input port, but most of service differentiation — buffer management and packet scheduling — is performed at the output port.
- There is a large difference between the search spaces of routing lookup and QoS lookup. The size of routing table is very large and ever-increasing with the growth of Internet, but the filtering rule set of QoS classification is small and remains stable.
- Conventional routing lookup is based solely on destination addresses, which is a one-dimensional search, but QoS lookup is based on multiple fields, which is a multi-dimensional search.

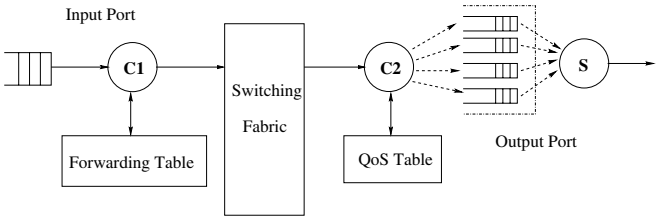

Figure 1: The architecture of the two-stage packet classifier at core routers

Note that at core routers, our QoS lookup is mainly restricted to the IP header fields, and the transport-layer information is accessed only if necessary. The proposed QoS lookup should not become the performance bottleneck inside the router. Moreover, the decoupling greatly simplifies the implementation of packet classification. The architecture of the two-stage packet classification is illustrated in Figure 1, where the forwarding table is the local version of routing table in the line card. With the forwarding table, the routing/switching decision can be made locally at each input port.

3.2 Fine-grained QoS classifier

As the key component of the sf-DiffServ architecture, the proposed QoS classifier at routers uses several fields in the IP header for QoS classification in addition to the DS field. Transport-layer information is extracted to further divide a BA into a UDP aggregate and a TCP aggregate, and then to distinguish TCP control segments that mainly consist of ACKs, from TCP data segments in the TCP aggregate. QoS classification can be modeled at three different hierarchical levels as shown in Figure 2.

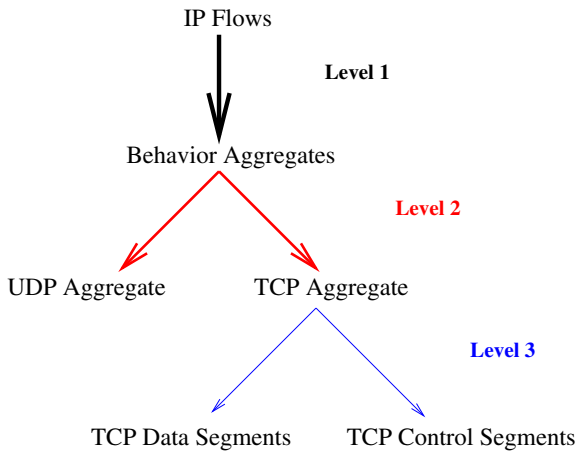

Figure 2: The three-level QoS classification

At the first two levels, it is straightforward to set the filtering rules. By checking DS and *protocol type* fields in the IP header against the filtering rules, the QoS classification is simple and does not cause any ambiguity. However, the accurate TCP control segment identification at level 3 could be much more complex. To implement the identification of TCP control segments at IP routers, the easiest way is to utilize one of the unused bits of DS field in the IP header. However, the problem with this solution is that it requires the modification of the IP header and cooperation from end-hosts. Moreover, the IETF has proposed using the two unused bits of DS field for the deployment of the Explicit Congestion Notification (ECN) mechanism [23] at routers.

So, we propose size- and port- based identification of TCP control segments without requiring any new bit. Note that in the following we present a detailed description of a general TCP control segment identification, which can be applied to the identification of each individual TCP control segment like SYN, FIN, ACK and RST. The only difference at the port-based version is to check different bits in the 6-bit flag field.

3.2.1 TCP Control Segment Identifications

Initially, each TCP segment is encapsulated in a single IP packet, but IP fragmentation could occur at intermediate routers. Once the IP packet is fragmented, only the first fragment contains the TCP header. So, the IP packet that contains the TCP header must have a zero fragmentation offset. By checking the *fragmentation offset* field in the IP header, the ambiguity caused by IP fragmentation is eliminated. Theoretically, if an IP packet encapsulates a TCP control segment, it should meet the following requirements: (1) its protocol type is TCP; (2) its fragmentation offset is zero; and (3) the corresponding flags in the TCP header are ON.

However, recent Internet traffic measurements [8, 31] have shown that about 40% of all IP packets are 40 bytes long, most of which are TCP control segments, implying that an overwhelming majority of TCP control segments are 40 bytes long. Then, based on whether the TCP header is accessed or not, we have two versions of TCP control segment identification: *lightweight* and *heavyweight*. Since IP options are included primarily for network testing or debugging, and the fraction of packets with IP header options are typically less than 0.003% [15], it is reasonable to assume that no IP option fields are attached to TCP control segments.

The lightweight version is a size-based classifier. It takes advantage of the above observations, and only checks the *total length* field in the IP header. The base filtering rule of TCP control segment identification is that the IP packets whose total length are 40 bytes, are classified as TCP control segments. The rationale behind this is that, since the IP header without options is 20 bytes and the TCP header without options is 20 bytes, a total of 40 bytes is the minimum size of an IP packet that encapsulated a TCP segment (without any payload). Considering TCP options — MSS option (4 bytes), Window scale factor option (3 bytes), Timestamp option (10 bytes) and Selective Acknowledgment option (10/18/26 bytes) — that can appear in the TCP control segments like SYNs or ACKs and

the requirement of a 4-byte boundary by padding NOP options, the complete filtering rule of TCP control segment identification is set as:

$$Rule: \{ X \mid X = 4 \cdot n;\ 10 \leq n \leq 20\}$$

where X is the total length of an IP packet and n is an integer. Since the maximum space that TCP options can use is 40 bytes, the maximum packet length of a TCP control segment is 80 bytes.

The main advantage of the lightweight version is that there is no need to access the TCP header, thus reducing overhead significantly. Its chief disadvantage is inaccuracy. The tiny TCP data segments that meet the filtering rules will be mis-classified as TCP control segments. However, because the tiny TCP data segments are most likely to belong to interactive TCP sessions, they have similar features of TCP control segments, i.e., small size and loss-sensitive TCP performance. For end-to-end TCP performance it is beneficial to separate them from, and give priority over, other TCP data segments. Moreover, the proposed adaptive weight-based resource manager has the ability to cope with this inaccuracy.

Since the lightweight version does not access the TCP header, it can not further differentiate TCP control segments into SYNs, FINs, ACKs and RSTs. To achieve accurate and fine-grained TCP control segment identification, the TCP header needs to be accessed. The heavyweight version of TCP control segment identification is a port-based classifier. The matching scope is outside the IP header, and hence, the TCP flags of the TCP header are checked. Besides the additional overhead in accessing the TCP header, IPSec makes the port-based classification difficult. However, a multi-layer IPSec protocol [40] has been proposed, which allows trusted routers to access the transport-layer information.

Even in the heavyweight version of TCP control segment identification, especially for ACK identification, we still need to rely on the *total length* field in the IP header to eliminate the ambiguity caused by the following two reasons:

- Some TCP implementations always set the ACK-flag bit ON once the TCP connection is established [29]. Furthermore, some malicious TCP senders could intentionally set TCP flag bit ON in its TCP data segments.
- The piggyback mechanism in which ACKs are sent along with a reverse-direction data flow, results in a packet that could be interpreted as a TCP data segment or TCP ACK.

The ambiguity caused by always-on ACK-flag and piggybacking can be solved by simply checking the total IP packet length. The large packet with ACK-flag ON is classified as a TCP data segment, but the small packet with ACK-flag ON is classified as a TCP ACK. Considering the addition of TCP option fields like the Timestamp in the TCP header, we set the threshold to 80 bytes, as the available bytes for TCP options is 40. If the total packet length is larger than 80 bytes, the packet is classified as a TCP data segment. Otherwise, it is classified as a TCP ACK.

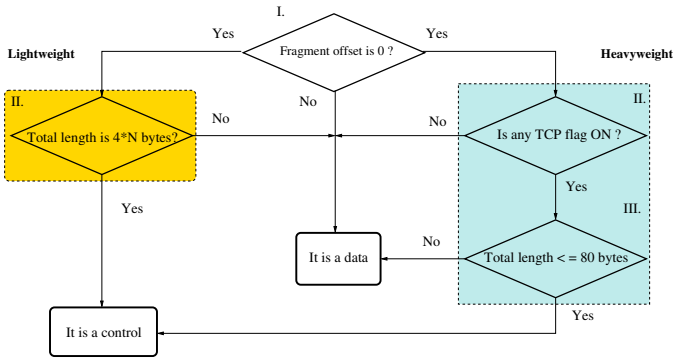

Figure 3: The flowchart of the TCP control segment identification algorithm

In summary, we suggest the lightweight version be used at core routers, and the heavyweight version be used at edge routers. A detailed description of the complete TCP control segment identification algorithm is given in Figure 3, where N is an integer such that $10 \leq N \leq 20$. There are three major steps in the heavyweight version, but only two steps in the lightweight version are given in Figure 3.

3.2.2 Validating lightweight identification

To validate the lightweight version of TCP control segment identification algorithm, six Internet traces taken at three different sites [21] are used. All the traces were collected between February 2002 and March 2002. The three chosen sites are located at high-bandwidth interconnection points. ADV represents the site where the OC3c (155Mbps) PoS (Packet over Sonet) link connects the Advanced Network and Services premises in Armonk, NY, to their ISP and the Internet2/Abilene network. ANL is referred to the OC3c link between the Argonne National Laboratory and the Ameritech Network Access Point (NAP) in Chicago. BUF is the site where the OC3 PoS link connects NYSERnet's router and University at Buffalo's router.

Traces	Mis-classified	Total Number	Error ratio
ADV-1	665	97053	0.68%
ADV-2	354	45463	0.78%
ANL-1	4351	560223	0.77%
ANL-2	2665	437085	0.61%
BUF-1	2218	208301	1.06%
BUF-2	2390	605127	0.39%

Table 1: Effect of mis-classification

From these traces, we found that no TCP control segment is mis-classified as a data segment and only an insignificant number of tiny TCP data segments are mis-classified as control segments. The filtering rule of 4-byte boundary significantly reduces the inclusion of tiny data segments (less than

80 bytes) in control segments. For example, in BUF-1 trace, without this rule, there would be 13318 tiny data segments that are mis-classified. However, after applying this rule, the number of mis-classified data segments is reduced to 2218. Table 1 gives the percentage of mis-classification of TCP data segments vs. the total data segments in each trace. Moreover, over 97% of the mis-classified tiny TCP data segments are with "PSH" flag ON in its TCP header.

3.2.3 Discussion

Currently, only a negligible part of the IP traffic is reported to belong to IPv6 [8, 31], so the proposed QoS classifier is based only on IPv4. Compared to IPv4, the most important changes in IPv6 lie in the packet format. The header format of IPv6 is very different from that of IPv4. However, the following two features of IPv6 will make the TCP control segment identification even easier and faster: (1) IPv4's variable-length options field is replaced by a series of fixed-format headers, and each IP packet has a base header followed by zero or more extension headers; (2) no fragmentation occurs at intermediate routers, and all the fragmentation and reassembly are restricted to end-hosts. So, it is easy to adjust the proposed QoS classifier to work properly in the context of IPv6.

The emergency of Internet Telephony — also called voice over Internet Protocol (VoIP) — does not pose a threat to the lightweight version TCP control segment identification, since VoIP data streams are carried by Real Time Transport Protocol (RTP) that is running on top of UDP, instead of TCP [24]. Moreover, the real audio streams show a significant regularity on packet lengths — concentrating on 244/254, 290/300 and 490/502 bytes [16], which are much larger than 80 bytes.

3.3 Adaptive weight-based resource manager

To enable better service differentiation and resource isolation between thinner aggregates, we propose an adaptive weight-based resource manager for IP routers. A hierarchical link-sharing structure is built, which is similar to the hierarchy of QoS classification. As shown in Figure 4, the root of the resource tree is the total link capacity. As the level of the resource tree gets lower, the IP flows that share the link are split into thinner aggregates. Each leaf node has its own queue, and every classified incoming packet is then inserted into the appropriate queue. Subsequently, the weighted round-robin scheduler will take care of these queued packets and select the next packet for transmission.

At level 1 of the resource tree, each node represents the allocated bandwidth to each BA. The bandwidth allocation and priority assignment at the BA level are done by the Bandwidth Broker (BB) [20] via Service Level Agreements (SLAs). At level 2 of the resource tree, the BA is divided further into a UDP aggregate and a TCP aggregate. Each node at level 2 corresponds to the bandwidth allocated to the thinner UDP/TCP aggregate. Within each BA, the UDP aggregate and the TCP aggregate have the same priority and a weighted

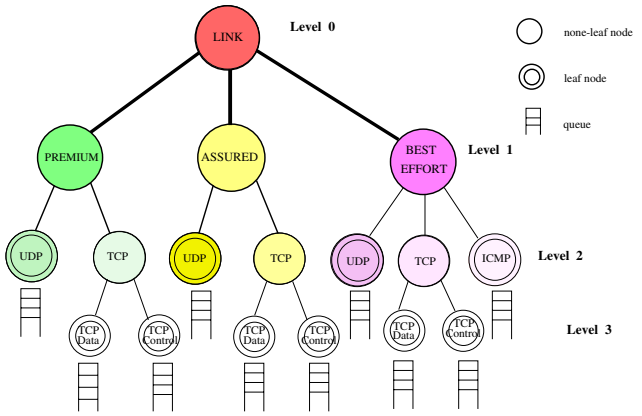

Figure 4: The link-sharing framework

round-robin scheduling scheme is used between them. The composition of each BA is different. Most of an EF (Expedited Forwarding) aggregate is UDP real-time audio/video. However, in the AF (Assured Forwarding) and BE (Best-Effort) aggregates, the majority of packets belong to TCP. The weights assigned to the thinner aggregates are based on the recent empirical studies reported in [8, 15, 16, 31], which are listed in Table 2. Here we assume the total weight of each BA is 1. The total weight assigned to UDP and TCP aggregates in BE is 0.99, since ICMP packets account for less than 1% of all IP packets and, by default, these ICMP packets are treated as BE traffic. Note that the weights at level 2 are tunable parameters that can be adjusted by network administrators to meet their local requirements. The weight assignment is strictly enforced only when there is no empty queue. Once a queue is empty, its assigned weight can be temporarily shared by other non-empty queues until the next packet enters the queue.

Type	Expedited	Assured	Best-Effort
UDP	0.7	0.05	0.04
TCP	0.3	0.95	0.95
ICMP	0	0	0.01
Total	1	1	1

Table 2: Level-2 weight distribution in different BAs

Each node at level 3 of the resource tree represents the bandwidth allocated to TCP data or control segments. The TCP data and control segments within the same TCP aggregate are also scheduled according to the weighted round-robin policy. The guiding principle for the weight setting at level 3 is that preference is given to control segments but there is a strict limit on the weight of control segments. The weight preference to control segments is the embodiment of resource overprovisioning as suggested in [22], and the strict limit prevents the misuse of preference. Because the overwhelming majority of the TCP control segments are TCP ACKs, we first present the rule of setting the weight of TCP ACKs, and then use the ACK weight as the baseline to derive the weights of

all TCP control segments.

3.3.1 Setting the weight of ACKs

The rule of thumb for setting the weight for TCP ACKs is to approximate the upper bound of their bandwidth consumption. If the weight is measured in number of packets, the ratio of the weight of TCP ACKs to that TCP data segments is 1:1. This *one data segment vs. one ACK* policy is based on the fact that the transmission of a TCP data segment will later trigger a TCP ACK. Considering wide deployment of the delayed ACK mechanism at TCP receivers, weights are assigned to give preference to TCP ACKs, and this preference is also intended to provide a safety cushion to other TCP control segments and tiny TCP data segments in case the lightweight version of TCP control segment identification is used.

Adhering to the policy of *one data segment vs. one ACK*, if the weight is measured in number of bytes, then we should consider the average packet size. As mentioned earlier, a great majority of TCP ACKs are 40 bytes long, so the average size of TCP ACK is 40 bytes. Common MSSs of TCP implementations are 512, 536 and 1460 bytes [8, 31]. Including the 40 bytes of both the IP and TCP headers, the total packet lengths for these MSSs are 552, 576, and 1500 bytes, respectively. Their observed ratio is 1:1:2. Thus, the average size of a TCP data segment is 1K bytes and the ratio of the weight of ACKs to that of data segments is $1 : 25$.

The traffic load distribution inside the DS domain will be balanced by routing algorithms and traffic engineering. At an interface of a core router, the volume of outgoing TCP data traffic equals that of incoming TCP data traffic, and each outgoing TCP data segment [2] implies an incoming TCP ACK to the interface. Therefore, this *one data segment vs. one ACK* policy is valid for core routers inside a DS domain.

However, at a leaf router or a boundary router between two different DS domains, this policy is often invalid due to the asymmetry of traffic load. This traffic load asymmetry has been observed in traffic measurements of the trans-Atlantic link [31]. The weight of TCP ACKs should depend on the weight allocated to the TCP data segments in the reverse direction. The weight of the ACK aggregate at leaf or boundary routers can be set based on traffic measurements or with the help of the Bandwidth Broker. Like the weight setting at level 2, the weight assigned to ACKs, which is the baseline of TCP control segments, is also a tunable parameter and can be adjusted locally.

Once the weight of the ACK aggregate is set, we use a simple adaptive calibration scheme to derive the weight of all TCP control segments for which the ACK weight is used as the baseline. The mechanism of the weight calibration works similarly to the adaptive-weighted packet scheduling of EF traffic [33]. Its goal is to increase the flexibility of the resource manager to absorb bursty control traffic and the tiny data segments that are mis-classified as TCP control segments.

[2] Or two outgoing TCP data segments if the delayed ACK mechanism is ON at the TCP receiver.

3.3.2 Adaptive weight calibration of TCP control aggregate

As in [33], we use the estimated average queue size of the TCP control aggregate to adaptively adjust the weight. The average queue size of the TCP control aggregate is calculated by using a low-pass filter with an exponentially-weighted moving average. Let *avg* be the average queue size, q the instantaneous queue size and f_l the parameter of the low-pass filter, then the average queue size of TCP control aggregate is estimated as:

$$avg \leftarrow (1 - f_l) \cdot avg + f_l \cdot q.$$

To reduce the instantaneous fluctuation of queue size, the parameter of the low-pass filter f_l is set to 0.01.

Assuming that the weight of TCP ACKs is w_a, we set the original weight of TCP control aggregate w_c to $1.2\ w_a$. To adaptively calibrate the weight of TCP control aggregate, two thresholds, min_{th} and max_{th}, are introduced. By keeping the average queue size of the TCP control aggregate below the maximum threshold, bursty losses of TCP control segments are prevented. To accomplish this, the weight of control aggregate should be proportionally increased once the average queue size of control aggregate exceeds the minimum threshold. The values of min_{th} and max_{th} are set to one fourth and three fourths of the buffer, respectively. The linear relationship between the weight and the average queue size of control aggregate is given by:

$$f(C) = \begin{cases} w_c, & C \in [0, min_{th}) \\ \frac{(U - w_c) \cdot (C - min_{th})}{max_{th} - min_{th}} + w_c, & C \in [min_{th}, max_{th}) \\ U, & C \in [max_{th}, full] \end{cases}$$

where $f(C)$ is the weight function of control aggregate, U is the upper limit that the weight of control aggregate can reach, and C is the average queue size of control aggregate. Since the total weight for TCP aggregates is fixed, the increase of control aggregate's weight must cause the same amount of decrease in the data aggregate's weight. However, once the average queue size of control aggregate reduces below max_{th}, the weights taken from data aggregate will be returned.

The weight calibration favors the control aggregate but disfavors the data aggregate, which is consistent with the guiding principle of the weight settings. The rationale behind this is that the bandwidth taken by the data aggregate is usually much more than the bandwidth consumed by the control aggregate; the small amount of bandwidth shift from the data aggregate to the control aggregate can prevent bursty losses of the control segments, but only leads to a single isolated data packet loss or just a longer queueing delay. However, the weight of control aggregate cannot exceed the upper limit, which prevents the abuse of preferential treatment of TCP control segments and protects the TCP data aggregate from starvation. U is set to $2w_c$ in our simulation.

4 Performance Evaluation

The proposed sf-DiffServ architecture is evaluated by simulation with *ns-2* [18, 32]. According to the purpose of simulation, we categorize the simulation experiments into two different classes. One is used for evaluating the capability of service differentiation, and the other is used for evaluating the capability of resource isolation under flooding attacks. To demonstrate the capability of providing better network QoS, we compare the received service of different class users in the *sf*-DiffServ, simple marking and the existing DiffServ architectures.

4.1 The simulation setup

The simulated network topology shown in Figure 5, is a relatively simple, yet sufficiently representative topology. All nodes are in a single DS domain. Each end-host is connected to its respective edge router, and the edge routers are connected via core routers. The link capacity and one-way propagation delay between an end-host and an edge router are 10 Mbps and 1 ms, respectively. The bandwidth and the link delay between an edge router and a core router are 3 Mbps and 8 ms, but those between two core routers are 1 Mbps and 16 ms. The UDP/TCP data segment size is set to 1000 bytes, and the TCP control segment size is set to 40 bytes. The version of TCP used in the simulation is TCP New-Reno since it has been widely deployed in the Internet. The initial TCP congestion window size is set to 2 and the delayed-ACK mechanism is ON. With respect to the direction of targeted TCP data flows, we name the path $R_3 \rightarrow R_1$ as the *forward* path and the path $R_1 \rightarrow R_3$ as the *backward* path.

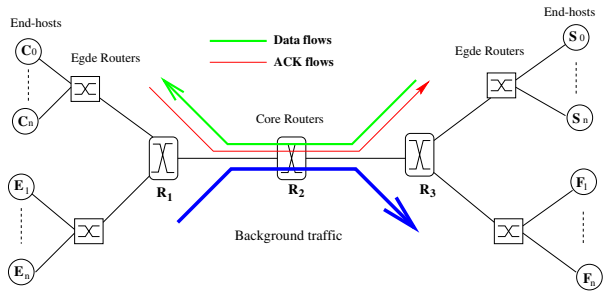

Figure 5: The network topology used for simulation

Note that, although the TCP version in our simulation experiment is New-Reno, most of the simulation results in this paper are applicable to all TCP variants for the following reasons. First, the TCP behaviors after a retransmission timeout for all of these schemes are similar. TCP variants differ only in the way of recovering from packet losses after a fast retransmit. Second, the ACK losses in the reverse path only lead to a timeout or slower congestion window growth, but cannot trigger a fast retransmit. Finally, in most cases of our simulation experiments, we intend to make the forward path congestion-free.

4.2 Service differentiation

We first focus on the short-lived TCP flows and measure the end-to-end latencies of different client requests experienced in the *sf*-DiffServ and the existing DiffServ architectures. Then, we study the long-lived TCP flows and measure the ACK loss rate and effective throughput, where we not only compare the sf-DiffServ with the existing DiffServ, but also with the marking scheme for TCP ACKs proposed in [22].

4.2.1 End-to-end latencies: short-lived TCP flows

In our simulation for evaluating end-to-end latencies, two clients C_0 and C_n on the same stub network send requests to a remote web server S_0. The two request-response transactions are performed by two TCP connections, which have the same RTT. The request sent by client C_0 is premium, and that by client C_n is basic. These two requests are simultaneously sent out. In response to each request, 20K bytes (i.e., 20 TCP data segments[3]) of data is transmitted from server S_0. The server's response to client C_0 receives premium service, but the one to client C_n is delivered with best-effort service. Background traffic, which consists of three TCP connections with large initial *ssthresh* of 64KB and a bursty UDP connection with Exponential ON/OFF, is sent from E_i to F_i, periodically congesting the backward path of the two TCP connections. S_0 supports service differentiation and isolation as in WebQoS [6]. A premium request is given priority over basic ones, and is guaranteed to get a faster response than the basic ones. Assume that the server is slightly overloaded by the incoming requests, and the time difference between the start of processing the premium and basic requests is 300 ms. The server's processing overhead for a premium request is similar to the one for a basic request, which is set to 5 ms.

As shown in Figure 5, in the *forward* path $R_3 \rightarrow R_1$, the server's responses are sent from S_0 to C_0 and C_n; but the ACKs of C_0 and C_n are delivered to S_0 in the *backward* path $R_1 \rightarrow R_3$. The background traffic also runs along the *backward* path. We assume that the forward path is lightly-loaded without any congestion. In contrast, the background traffic causes periodic congestions in the backward path. Due to the burstiness and the global synchronization [38] of the background traffic, the state of the backward path fluctuates wildly between "congested" and "idle." It is not uncommon that C_0's ACKs are dropped during the congestion in the backward path, but C_n's ACKs — which occur later in time during the idle period of the backward path — are delivered quickly.

In the current DiffServ architecture, which does not support service differentiation among ACK flows, the ACK losses of a premium-class flow in the backward path could undo or degrade the service differentiation achieved by the server and the forward path. The simulation results of the current DiffServ architecture are plotted in Figure 6 (Premium I, II, III and Basic), clearly showing this symptom. According to the

[3] The Internet traffic measurements show that the average number of packets per TCP flow ranges from 16 to 20.

impact of the ACK losses of a premium session upon its TCP performance, the simulation results are categorized into three different cases.

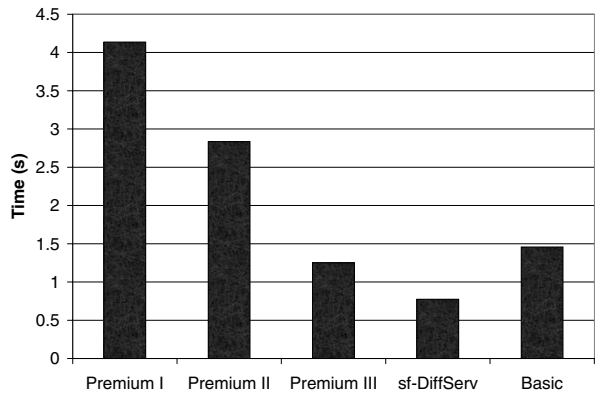

Figure 6: The response times of premium and basic requests

Case 1. ACKs in the initial congestion window are dropped during the first RTT, which results in a retransmission timeout. If the initial congestion window size is 1, a single ACK loss leads to the retransmission timeout. The lethal effect of losing self-clocking at the very beginning of TCP transmission lies in that the initial retransmission timeout is set to 3 seconds. Being idle for 3 seconds is a big loss for the premium session, which is shown in Figure 6 (Premium I).

Case 2. bursty ACK losses occur when the congestion window size is small, and result in a retransmission timeout. However, the ACK losses must happen after the first RTT. The retransmission timeout has been correctly set, instead of a large default value. Even so, the occurrence of retransmission timeouts still greatly degrades TCP performance as shown in Figure 6 (Premium II).

Case 3. bursty ACK losses occur when the congestion window size is large enough to prevent retransmission timeouts. ACK losses only slow down the growth of congestion window size and increase the burstiness of data transmission in the forward path. The TCP performance degradation shown in Figure 6 (Premium III) is not as significant as those caused by retransmission timeouts.

The same simulation experiments are conducted for the sf-DiffServ architecture, where the lightweight version of TCP control segment identification is used. Due to finer traffic classification and service differentiation among ACK aggregates, the ACKs of the premium session are guaranteed to receive better differentiated service than those of the basic session. Moreover, because separate buffers are used for queueing ACKs and data segments, the queueing delay experienced by the ACKs is reduced. The simulation result is shown in Figure 6 (sf-DiffServ). It clearly shows the improvement of response time of the premium session in the sf-DiffServ over the uni-directional counterpart.

4.2.2 ACK loss rate and goodput: long-lived TCP flows

The experimental configuration is somewhat different from the one used above. Three targeted TCP connections are established from S_1 to C_1, which receive premium, assured and best-effort services, respectively. In addition to the three targeted TCP connections, two more TCP connections carry best-effort data from S_2 to C_2. All of them have infinite amounts of data to send. The resources along the forward path $R_3 \rightarrow R_1$ is properly provisioned for the premium and assured traffic, but the remaining network resources are periodically exhausted by the best-effort traffic, causing random data losses to occur in the forward path.

On the backward path $R_1 \rightarrow R_3$, similar background traffic is generated between E_i to F_i, and shares the same path with the targeted ACK flows. The background traffic is a mixture of premium, assured and best-effort traffic. Compared to the simulation configuration in the forward path, there are two key differences in the backward path:

- the network resources for the premium and assured services in the background traffic are under-provisioned; and
- the best-effort traffic consists of not only TCP flows but also UDP flows, which causes severe congestion in the backward path thus resulting in bursty packet losses.

The ACK loss rates of targeted TCP connections are charted in Figure 7, where *Marking* refers to the ACK marking scheme proposed in [22], and *Existing* refers to the current DiffServ architecture. The effective throughput of the targeted TCP connections are shown in Figure 8, where *Reserved* refers to the reserved bandwidth for EF and AF flows. The simulation results show that:

- sf-DiffServ provides better service isolation for ACK flows, significantly lowering the ACK loss rate and increasing effective throughput; and
- the ACK marking scheme cannot support service isolation for ACK flows when network resources are under-provisioned, thus resulting in bursty ACK losses and hence degrading TCP performance significantly.

For EF and AF traffic, the ACK loss rates of *Marking* are much lower than those of *Existing*, but are much higher than those of *sf-DiffServ*. Moreover, most of ACK losses are bursty rather than random, lowering effective throughput. For EF traffic, the main reasons for bursty ACK losses are: (1) the buffer space for premium service is very small, and can only accommodate 1 or 2 data packets; (2) the size of a data segment is much larger than that of an ACK. Once the buffer has been filled with data segments, all the incoming ACKs will be dropped.

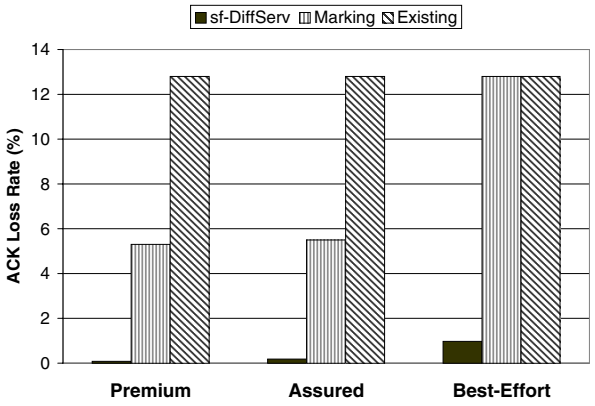

Figure 7: The ACK loss rate in different DiffServ architectures

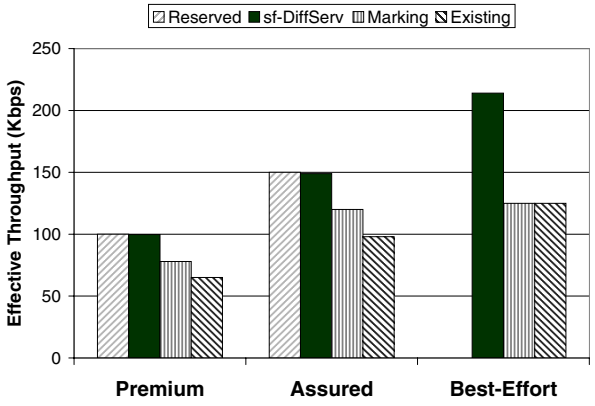

Figure 8: The effective throughput in different DiffServ architectures

In the *Existing* and *Marking* DiffServ architectures, AF traffic share the same FIFO queue with best-effort traffic, but AF packets are much less likely to be dropped than best-effort ones. However, without proper resource provisioning for ACK flows, the ACKs are more likely to be marked as high drop-precedence packets at edge routers due to the corresponding traffic profile violation. Under a severe congestion, all the packets marked with high drop-precedence will be dropped, causing bursty ACK losses. Since bursty ACK losses cause much severer degradation to TCP performance than random ACK losses, even modest ACK loss rates for EF and AF can greatly reduce their effective throughput.

Compared to the *Existing* DiffServ, the ACK marking scheme provides no improvement to best-effort traffic. Best-effort ACKs experience high loss rates in the backward path because of the congestion caused by the UDP flows in the background traffic. Furthermore, due to data losses in the forward path, an ACK loss for retransmission in the backward path leads to a timeout, reducing $cwnd$ to 1, triggering a slow-start, and hence, degrading effective throughput significantly.

In contrast, the sf-DiffServ architecture significantly improves the performance of best-effort TCP traffic, thanks to its resource isolation between UDP and TCP flows, as well as between ACKs and TCP data segments within the best-effort class. The sf-DiffServ not only provides better differentiated service to high-tiered services, but also significantly improves the performance of best-effort TCP sessions.

4.3 Resource Isolation

sf-DiffServ also provides a built-in protection mechanism to counter DDoS attacks. The layer-4 traffic splitting greatly reduces the performance degradation caused by DDoS attacks like UDP flooding, ICMP flooding (i.e., smurf), TCP SYN flooding and ACK flooding. The resource isolation provided inside the BE traffic class is especially valuable, since the edge routers in the DiffServ architecture perform traffic conditioning and policing on EF and AF traffic, but not on BE traffic. Most importantly, most of the flooding traffic will be dropped by the first few routers before they reach the core of the network, thus limiting the damage caused by the flooding source mainly to the local stub network where it originated. The cascaded throttling of flooding traffic at the first few routers makes the rest of the Internet unaffected, and saves the network bandwidth.

In our flooding experiments, there are 10 flooding sources in each stub network except for the one that the victim F_1 belongs to. The flooding rate at each source is constant and set to 5000 packets per second. At the same time, there are 10 TCP connections running from E_i to F_i, where i is an integer in [1,10], carried by the BE service as the normal background traffic. We first measure the volume of the flooding traffic that reaches at the victim under different DiffServ architectures. Four types of flooding attacks — SYN, ACK, UDP, and ICMP flooding — are simulated. All the flooding traffic is transported by the BE service. Since there is no difference in treating the BE traffic in the *Existing* and the *Marking* architectures, we normalize the various flooding traffic reached the victim in these architectures to 1 to make the presentation easier. Then, the flooding traffic received at the victim in the *sf-DiffServ* architecture is properly scaled based on the normalization. Figure 9 shows that *sf-DiffServ* throttles the flooding volume that reaches the victim and effectively protects the victim from flooding attacks. Moreover, during flooding attacks, the TCP effective throughputs in *Existing* and *Marking* are reduced almost to zero, but the one in *sf-DiffServ* can still achieve 95% (in the cases of UDP and ICMP flooding) and 85% (in the case of SYN and ACK flooding) of the bandwidth assigned to the entire BE traffic, thanks to the layer-4 resource isolation.

For EF or AF traffic, the UDP and ICMP flooding do not cause much damage in all DiffServ architectures because the edge routers perform traffic conditioning and policing on EF and AF traffic. However, the *Marking* DiffServ exposes more vulnerability to the ACK flooding attacks. In this architecture, the ACK flows are accepted without strict policing based

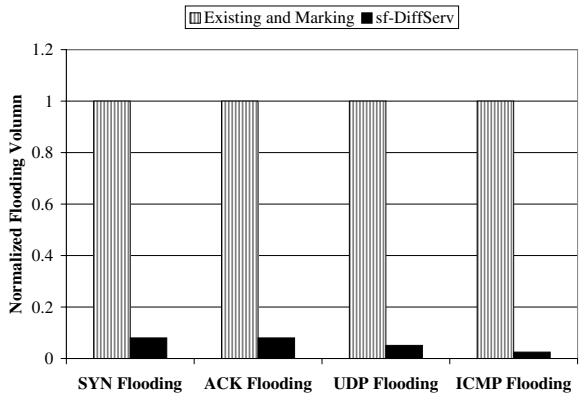

Figure 9: The flooding volume received at the victim

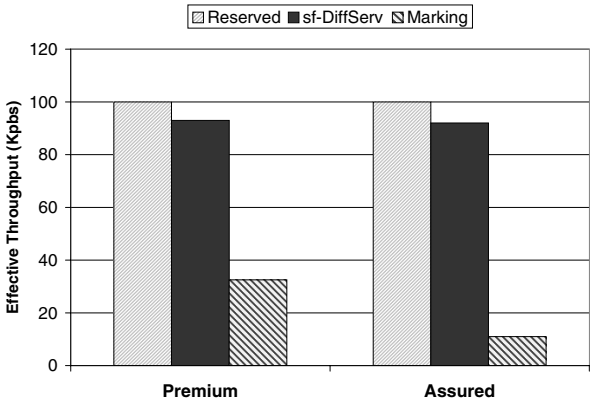

Figure 10: The goodput of conformant EF and AF flows under the ACK flooding attack

on the belief that the small bandwidth requirement by ACK flows can be absorbed by over-provisioning. The flooding ACK flows marked as EF or AF traffic can seriously violate the traffic profile between the stub network and the leaf router that connects the stub network to the Internet. Even worse, in large-scale DDoS attacks, even if only a small number of ACKs are flooded from each attacking source, once these ACKs are aggregated at core routers where no traffic conditioning is performed, the flooding ACK aggregates can "steal" the reserved bandwidth from the conformant aggregates. Figure 10 charts the goodputs of conformant EF and AF flows from C_1 to F_1 in *Marking* and *sf-DiffServ*, and clearly shows the vulnerability of *Marking* to the ACK flooding attack and the robustness of *sf-DiffServ* to the same attack. Note that the conformant EF flow is carried by UDP, but the AF one is carried by TCP.

5 Conclusions

We presented a scalable fine-grained DiffServ architecture to provide layer-4 service differentiation and resource isolation. The key components of the sf-DiffServ architecture are the fine-grained QoS classifier and the adaptive weight-based resource manager. A two-stage packet classification mechanism is devised to decouple the fine-grained QoS lookup from the routing lookup at core routers. BAs are further divided into thinner aggregates. By using separate queues and adaptive-weighted bandwidth allocation, better service differentiation and isolation are achieved for these thinner aggregates. No bi-directional resource reservation is required.

We evaluated the performance of the sf-DiffServ architecture by simulation. The simulation results show that:

- sf-DiffServ guarantees that high-tiered TCP sessions receive better service and hence yield better performance in terms of loss rate, end-to-end delay and effective throughput, than low-tiered TCP sessions;
- It not only achieves better differentiated service for high-tiered services, but also significantly improves the performance of best-effort TCP sessions;
- It provides a built-in protection mechanism to counter DDoS attacks, especially UDP and ICMP flooding.

Furthermore, the simulation results demonstrate that a simple ACK marking scheme does not provide good service differentiation and isolation for ACK flows when network resources are under-provisioned. It exposes the vulnerability of EF and AF traffic to the ACK flooding attacks. The scalable fine-grained DiffServ architecture is therefore necessary to provide better network QoS to TCP sessions, and a simple but powerful built-in protection mechanism to counter DDoS attacks.

References

[1] L. Abeni and G. Buttazzo, "Hierarchical QoS Management for Time Sensitive Applications", *Proceedings of IEEE RTAS'2001*, Taipei, Taiwan, May 2001.

[2] T. Abdelzaher and K. G. Shin, "End-host Architecture for QoS-Adaptive Communication", *Proceedings of IEEE RTAS'1998*, Denver, CO, June 1998.

[3] G. Banga, P. Druschel and J. Mogul, "Resource containers: A new facility for resource management in server systems", *Proceedings of 3th Symposium on Operating System Design and Implementation*, New Orleans, LA, February 1999.

[4] H. Balakrishnan, V. Padmanabhan, and R. H. Katz, "The Effects of Asymmetry on TCP Performance" *Proceedings of ACM/IEEE MOBICOM'97*, Budapest, Hungary, September 1997.

[5] Y. Bernet *et al.*, "A Framework for Differentiated Services", *IETF Internet Draft*, February 1999.

[6] N. Bhatti and R. Friedrich, "Web Server Support for Tiered Services", *IEEE Network*, Vol. 13, No. 5, September/October 1999

[7] S. Blake *et al.*, "An Architecture for Differentiated Services", *RFC 2475*, December 1998.

[8] CAIDA's Traffic Workload Overview, http://www.caida.org/outreach/resources/learn/ traffic-workload/tcpudp.xml

[9] S. Floyd and V. Jacobson, "Link-sharing and Resource Management Models for Packet Networks" *IEEE/ACM Transactions on Networking*, Vol. 3, No. 4, August 1995.

[10] L. Garber, "Denial-of-Service Attack Rip the Internet", *Computer*, April 2000.

[11] P. Gupta and N. McKeown, "Packet Classification on Multiple Fields", *Proceedings of ACM SIGCOMM'99*, Cambridge, MA, September 1999.

[12] V. Jacobson, "Congestion Avoidance and Control", *Proceedings of ACM SIGCOMM'88*, Stanford, CA, August 1988.

[13] T.V. Lakshman and D. Stiliadis, "High Speed Policy-based Packet Forwarding Using Efficient Multi-dimensional Range Matching", *Proceedings of ACM SIGCOMM'98*, Vancouver, Canada, September 1998.

[14] J. McQuillan, "Layer 4 Switching", *Data Communications*, October, 1997.

[15] S. McCreary and K. Claffy, "Trends in Wide Area IP Traffic Patterns — A View from Ames Internet Exchange", *Proceedings of ITC'2000*, Monterey, CA, September 2000.

[16] A. Mena and J. Heidemann, "An Empirical Study of Real Audio Traffic", *Proceedings of IEEE INFOCOM'2000*, Tel Aviv, Israel, March 2000.

[17] A. Miyoshi and R. Rajkumar, "Protecting Resources with Resource Control Lists", *Proceedings of IEEE RTAS'2001*, Taipei, Taiwan, May 2001.

[18] S. Murphy, "DiffServ Additions to ns-2", May 2000, http://www.teltec.duc.ie/ murphys/ns-work/diffserv

[19] A. K. Mok, X. Feng, and D. Chen, "Resource Partition for Real-time Systems", *Proceedings of IEEE RTAS'2001*, Taipei, Taiwan, May 2001.

[20] K. Nichols, V. Jacobson, and L. Zhang, "A Two-bit Differentiated Services Architecture for the Internet", *RFC 2638*, July 1999.

[21] NLANR Network Traffic Packet Header Traces, http://pma.nlanr.net/Traces/

[22] K. Papagiannaki, P. Thiran, J. Crowcroft and C. Diot, "Preferential Treatment of Acknowledgment Packets in a Differentiated Services Network", *Proceedings of IWQoS'2001*, Karlsruhe, Germany, June 2001.

[23] K.K. Ramakrishnan and S. Floyd, "A Proposal to add Explicit Congestion Notification (ECN) to IP", *RFC 2481*, January 1999

[24] D. Rizzetto and C. Catania, "A Voice over IP Service Architecture for Integrated Communications", *IEEE Network*, Vol. 13, No. 3, June 1999.

[25] D. Rosu, K. Schwan, S. Yalamanchili, and R. Jha, "On Adaptive Resource Allocation in Complex Real-time System" *Proceedings of IEEE RTSS'97*, San Francisco, CA, December 1997.

[26] H. Schulzrinne, A. Rao, and R. Lanphier, "Real Time Streaming Protocol (RTSP)", *RFC 2326*, April 1998.

[27] O. Spatscheck and L. Peterson, "Defending Against Denial of Service Attacks in Scout", *3th Symposium on Operating System Design and Implementation*, New Orleans, LA, February 1999.

[28] V. Srinivasan, G. Varghese, S. Suri, M. Waldvogel, "Fast and Scalable Layer Four Switching", *Proceedings of ACM SIGCOMM'98*, Vancouver, Canada, September 1998.

[29] W. Stevens, *TCP/IP Illustrated*, Volume 1. Addison-Wesley Publishing Company, 1994.

[30] A. Striegel and G. Manimaran, "Dynamic Class-based Queue Management for Scalable Media Servers", *Proceedings of IEEE RTAS'2000*, Washington D.C, May 2000.

[31] K. Thompson, G. J. Miller, and R. Wilder, "Wide-Area Internet Traffic Patterns and Characteristics", *IEEE Network*, Vol. 11, No. 6, November/December 1997.

[32] UCB/LBNL/VINT, "Network Simulator", *ns-2*, 1999. http://www.isi.edu/nsnam/ns/

[33] H. Wang, C. Shen and K. G. Shin, "Adaptive-Weighted Packet Scheduling for Premium Service", *Proceedings of IEEE International Conference on Communications'2001*, Helsinki, Finland, June 2001.

[34] H. Wang and K. G. Shin, "Robust Congestion Recovery", *Proceedings of IEEE ICDCS'2001*, Phoenix, AZ, April 2001.

[35] P. Wang, Y. Yemini, D. Florissi, J. Zinky and P. Florissi, "Experimental QoS Performances of Multimedia Applications", *Proceedings of IEEE INFOCOM'2000*, Tel Aviv, Israel, March 2000.

[36] L. Welch *et al.*, "Adaptive QoS and Resource Management using a *Posteriori* Workload Characterizations", *IEEE RTAS'1999*, Vancouver, Canada, June 1999.

[37] R. West and K. Schwan, "Quality Events: A Flexible Mechanism for Quality of Service Management", *Proceedings of IEEE RTAS'2001*, Taipei, Taiwan, May 2001.

[38] L. Zhang, S. Shenker, and D. Clark, "Observations on the Dynamics of a Congestion Control Algorithm: The Effects of Two Way Traffic", *Proceedings of ACM SIGCOMM'91*, Zürich, Switzerland, September 1991

[39] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, "RSVP: A New Resource ReSerVation Protocol", *IEEE Network*, Vol. 7, No. 4, September/October 1993.

[40] Y. Zhang and B. Singh, "A Multi-layer IPsec Protocol", *Proceedings of 9th USENIX Security Symposium*, Denver, Colorado, August 2000.