Off-the-path Flow Handling Mechanism for High-Speed and Programmable Traffic Management

Hideyuki Shimonishi
System Platforms Research Laboratories, NEC Corporation
1753 Shimonumabe, Nakahara, Kawasaki, Kanagawa 211-8666, Japan
+81-44-396-3491
h-shimonishi@cd.jp.nec.com

Takashi Yoshikawa
System Platforms Research Laboratories, NEC Corporation
1753 Shimonumabe, Nakahara, Kawasaki, Kanagawa 211-8666, Japan
+81-44-396-2391
yoshikawa@cd.jp.nec.com

Atsushi Iwata
System Platforms Research Laboratories, NEC Corporation
1753 Shimonumabe, Nakahara, Kawasaki, Kanagawa 211-8666, Japan
+81-44-396-2744
a-iwata@ah.jp.nec.com

ABSTRACT

In this paper, we propose a high-speed and programmable traffic management mechanism to enable easy and timely innovations. A control framework introduced by 4D, Tesseract, or OpenFlow, separates control functions from the switch nodes to a control server so that a variety of network control policies can be implemented outside of the switches. Within this framework, we propose a mechanism to enable flexible flow-based traffic management so that a variety of innovative traffic management schemes can be realized. Per-flow traffic management, however, requires packet-by-packet state updates, which can spoil this control framework. The proposed mechanism consists of a control server that monitors traffic conditions using sampled packets sent from the switches and calculates per-flow packet discarding rate, and switches that discard incoming packets according to the discarding rate. Packet sampling and discarding do not require packet-by-packet state handling at the switches and thus allows controls from a control server. We also propose a mechanism to compress the discarding information using a time series of bloom filters, so that frequent control updates are allowed. We tested the mechanism with per-flow WFQ emulation and the simulation results showed very good per-flow fairness. Furthermore, we found that the flow table is compressed 600 times smaller and that the processing cost at the server and the switches is small enough for use with 10 Gbps links.

Categories and Subject Descriptors

C.2.3 [**COMPUTER-COMMUNICATION NETWORKS**]: Network Operationss – *Network monitoring, Network management.*

General Terms: Algorithms, Performance, Design

Keywords: Traffic management, QoS, programmable, high-speed, scalable, per-flow, bloom filter, network virtualization

1. INTRODUCTION

Several research initiatives including FIND [1], 4WARD [2], and Trilogy [3] have been started and much research has been conducted on "Future Internet". One of the most important features of the future Internet would be flexibility of network functions for easy and timely innovations. To accelerate the development of such innovations, programmable network testbed projects, such as GENI [4], have been discussed in order to facilitate innovative experiments in more realistic network environments. A control framework introduced by 4D [5], Tesseract [6], Ethane [13], or OpenFlow [7], separates control functions from the switch nodes to a control server so that a variety of network control policies can be implemented outside of the switches. For example, new routing policies or security policies can easily be tested without the need for introducing new functionalities into the switches.

Within this framework, we propose a mechanism to enable flexible flow-based traffic management. For example, new algorithms regarding active queue managements, DoS mitigation schemes, per-flow WFQ schedulers, to mention a few, are realized by control server programs. Per-flow traffic management, however, requires packet-by-packet state updates and thus requires fat and unaggregatable flow tables as well as rich functionalities on the switches. Considering that the number of flows can be as large as 1 million and their link speed is exceeding 10 Gbps in the Internet backbone links, realizing both programmability and high-performance at the same time is a big challenge. To achieve high-speed packet processing, switches should be stateless and its flow table size should be minimized. And, to control traffic management from a control server, control overhead has to be minimized so that frequent control updates are allowed.

A variety of per-flow bandwidth controls can be realized by adaptive packet discarding, which dynamically changes discarding rate according to control policies and traffic conditions, therefore the switches can be stateless. For example, active buffer management schemes, including Fair RED (FRED) [9] or Random Early Discard with Preferential Dropping (RED-PD) [10] are typical examples of per-flow discarding rate control. Similarly, DoS attack mitigation schemes can be within this framework of discarding specific messages of specific flows. For example, SYN-flooding attacks can be mitigated by selectively discarding SYN packets of specific flows. In addition, as it is shown in the following sections, per-flow bandwidth control using deficit round robin (DRR) [11] or weighted fair queue (WFQ) can also be emulated. Although delay or jitter control cannot be performed by merely discarding packets, the number of packets output, i.e. per-flow throughput, can be controlled. Also in [8], fair queuing is emulated in core switches without per-flow state maintenance, although it requires a packet header extension as well as per-flow state maintenance at edge switches.

To minimize control overhead between switches and control servers, as well as to compress the flow table at switches, we propose a flow handling mechanism that uses a time series of bloom filters [12]. One good feature of a bloom filter is that the filter only needs to include the flows under control, and the filter size is independent of the number of entire flows. For example, let us assume a case where there are 1 million flows in total and 10% of them are needed to be shaped, the filter size is a magnitude of 100 thousand, rather than 1 million. In fact, flows that send packets at a very high rate, or flows that attack the network, is only a small fraction of entire flows [10]. In addition, the flow table does not require a fine grain value for the discarding rate if the values are frequently updated (because the discarding rate is itself statistical). This is the similar case with a 1-bit D/A converter that generates an analogue signal (i.e. fine grain discarding rate) from a time series of "0" (i.e. not-discarding) and "1" (i.e. discarding). While the control server maintains multiple bloom filters to store multi-bit values for the discarding rate, the switch needs to keep one of the filters, which is frequently replaced with other filters.

Based on the above, we have developed a traffic management mechanism that takes advantage of both programmability at a control server and high-speed packet processing of switch hardware. A control server program monitors per-flow traffic conditions through packet sampling and maintains per-flow state information to calculate per-flow packet discarding rates according to the control policies. Since the server operation is basically activated upon a reception of a sampled packet, whose sampling rate is typically quite low, e.g., 1 packet out of 100, the server's processing cost should not be a problem. Then, the calculated discarding rate is sent to the switches, which simply discard packets according to this rate. Therefore, simple and fast packet handling is enough for the switches.

In the following, we outline the proposed mechanism and evaluate it with one typical application; per-flow packet scheduling emulation using WFQ. Actually, WFQ emulation is a challenging application to the proposed mechanism since it requires very strict state maintenance and thus larger control error could happen when there are a large number of short-lived flows, which can not be captured by sampling. On the other hand, applications like regulating heavy hitters or mitigating DoS attack packets are very good application. Nevertheless, we tested WFQ emulation to evaluate the ability of the mechanism.

2. Description of the Proposed Mechanism

2.1 Outline of the mechanism

Figure 1 illustrates the developed mechanism, which consists of a control server and switches, and Fig. 2 shows the switch functions and server functions. The control server can be a separate server machine, as shown in the figure, or can be located inside the switches as a local controller.

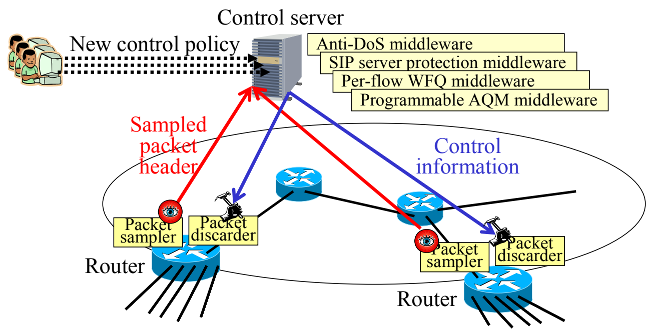

Fig. 1: Basic idea of the proposed mechanism

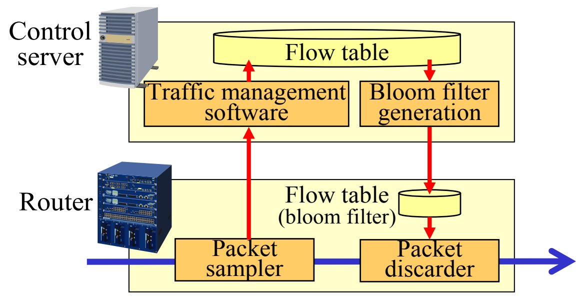

Fig. 2: Basic functions of the proposed mechanism

The server can control multiple switches, as shown in the figure, for network wide traffic management. With network wide traffic information, the server can make appropriate control decisions and instruct appropriate switches. For example, if the server can determine a switch that is close to the origin of a flow, the server only needs to control that particular switch. Alternatively, multiple servers can control a single switch so that, for example, the switch accommodates multiple virtual network slices and each slice has its own control servers for better maintenance or safety.

The server may use Virtual Machine technologies or User Mode Linux technology so that it maintains multiple control programs easily and safely. Control programs should be dynamically downloaded to the servers so that users can easily install new algorithms. Furthermore, if the switch accommodates multiple virtual networks or multiple types of flows, each of them should be dynamically configured in order to be bound with appropriate set of control programs. We will study there features as we develop a prototype system by carefully following related researches including GENI, PlanetLab [14], and VINI.

2.2 Switch function (packet sampling)

The switches, which can be edge switches, core switches, switches, firewalls, or any kind of network node, should sample packets and send their headers to the server(s). Standard mechanisms like sFlow [15] can be used for this purpose; therefore, there is no need to develop new mechanisms. A simple random sampling is enough for rate measurement, so there is no need to keep flow information or state information at the switches. Of course, it might be useful if the sampler can change the sampling rate according to the types of traffic, or collects certain types of control messages, e.g. TCP-SYN, SIP Invite, for more sophisticated traffic control applications.

2.3 Server function

A control program on a control server maintains per-flow state information. It updates flow status using the information carried on sampled packet headers and performs any operations needed for traffic management. When it determines the per-flow packet discarding rate or any other control information, it is compressed into bloom filters, which are sent to the switches periodically.

2.4 Switch function (packet discarding)

When a switch receives one of the bloom filters, it keeps the filter as a flow table. When it receives a packet on a line interface, it examines whether the flow the packet belongs to is registered on the filter or not, and if it is, the switch performs the predetermined operation, i.e. discards the packet at the indicated discarding rate, otherwise the switch passes through the packet.

2.5 Compression of flow table using bloom filters

Bloom filters are used for an efficient data structure for a flow table. They express whether an element (i.e. a flow) of a data set (i.e. set of flows going through a switch) is included in a subset of the elements (i.e. a set of flows that is indicated as needing to be controlled). Bloom filter F is a series of bits having pre-determined length N. To register a new flow whose ID is X in F, multiple bit positions of F are set according to hash functions $H_j(X)$ ($0<j\leq K$ and $0\leq H_j(X)<N$), where K is the number of hash functions. To check whether a flow is included in the filter or nor, bit positions of F indicated by the same hash functions $H_j(X)$ are examined. If all the bit positions are set, the flow is included otherwise the flow is not included. It is ensured that if a flow is included in the subset, the filter must indicate the flow is included, which means there are no false negatives. But, there is certain possibility of a false positive. We will discuss this possibility in the following section.

Since a bloom filter only states that a flow is IN or that a flow is OUT, multiple bloom filters are necessary to express fine grain control values. It would be possible to express 128 levels of a discarding rate using 7 bloom filters, but this means that the flow table requires 7 bloom filters to be kept on a switch. To reduce the flow table size, we also proposed periodically changing the bloom filter kept in a switch. As shown in Fig. 3, a control server maintains M bloom filters for M levels of a packet discarding rate. The server has a much larger memory than a switch, so the number should not be a problem. If a flow has a non-zero packet loss rate a, the flow is registered into aM randomly chosen bloom filters. Then, one of the filters is sent to the switch in a round robin manner. When a switch receives a new bloom filter, it replaces the one it had by the one it receives. Thus, packet loss rate varies over time at the switch, and if the filter update is generally adequate, the resulting packet loss rate becomes sufficiently smooth.

2.6 Application examples of the mechanism

There are a number of applications for our mechanism. We describe some of them below. A useful feature of a bloom filter is that key length can be arbitrary. Thus, any key, whether it is, for example, a flow ID, source or destination IP address, protocol ID, or message ID can be stored in the same filter. This makes our

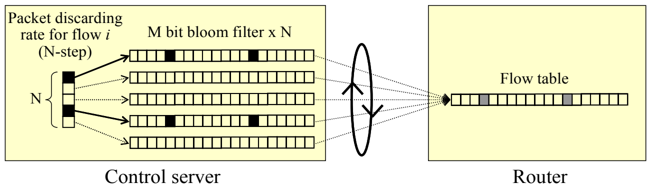

Fig. 3: compression of flow table using bloom filter

mechanism so flexible that many applications can be applied simultaneously

Active queue management (AQM)

Basically any type of AQM scheme can be written as a program on a server. AQM that differentiates per-flow packet loss rates can be performed using the bloom filter mechanism. Also basic AQM schemes, which use the same packet loss rate for all flows, can also be applied by filing bloom filters with "1".

Emulation of per-flow packet scheduler

Per-flow bandwidth control using a per-flow packet scheduler like WFQ or DRR can be emulated. By properly configuring packet discarder in a switch, a definition of "flow" can be flexibly configured. For example, be single TCP/IP flow or it can be a group of flows destined for the same IP destination.

DoS mitigation

A server program can detect DoS attacks by monitoring certain messages in the sampled packets and then configure the bloom filters in order to regulate a certain portion of the messages to mitigate the attack. When a server controls multiple switches, it may send control information to the switches close to the source of the attack or the switches close to the victim.

Call regulation

During disasters, SIP server failures, or congestion the mechanism can be used to regulate calls by probabilistically regulating SIP messages.

3. Application to WFQ Emulation

In this section, we describe per-flow WFQ emulation as one of the applications of a control server. As illustrated in Fig. 4, when a sampled packet header arrives at the server, following (1) – (3) operations are performed. (1) The server monitors flow behaviors using sampled packets and calculates per-flow queue length of a corresponding WFQ scheduler. The queue length is not a real one but rather it is a virtual queue length calculated as if there were a WFQ scheduler. (2) Packet discarding rate is calculated using an AQM mechanism and (3) the rate is encoded into bloom filters. Then, periodically, independently of a reception of a sampled header, one of the filters is picked up and sent to the switch. The switch is configured for appropriate definition of "flow", which could be TCP/UDP flow, or MPLS flow, or whatever, and discards packets as instructed by the server. Therefore, it is expected that the number of packets sent and discarded, and thus per-flow throughput, of both proposed mechanism and real WFQ scheduler is consistent.

3.1 Maintaining "virtual" queue

When the server receives a sampled packet header of a flow, it updates the length of a queue allocated to the flow by calculating incoming bytes to the queue and outgoing bytes from the queue, as follows;

$$Q_{i,j} = max(Q_{i,j-1} + I_{i,j} - O_{i,j}, 0), \tag{1}$$

where $Q_{i,j}$ is the queue length of flow i at the reception of j-th

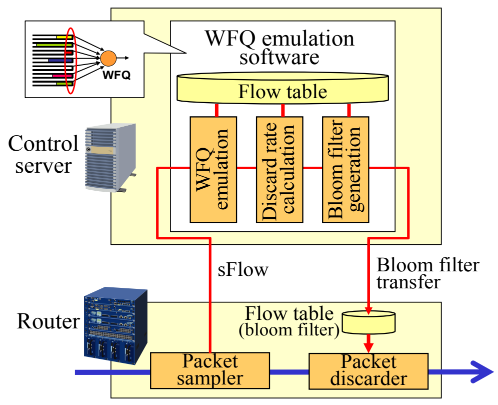

Fig. 4: example; WFQ emulation

sampled header. $I_{i,j}$ and $O_{i,j}$ are the incoming and outgoing bytes during the reception of $(j-1)$-th and j-th sampled headers, respectively

Regarding UDP flows, incoming bytes $I_{i,j}$ is estimated by

$$I_{i,j} = p\ SS, \tag{2}$$

where p is the sampling rate and SS is the segment size of a packet. Regarding TCP flows, $I_{i,j}$ can be obtained more accurately using TCP sequence numbers, as follows;

$$I_{i,j} = SN_{i,j} - SN_{i,j-1}, \tag{3}$$

where $SN_{i,j}$ is the TCP sequence number shown in j-th sample.

Outgoing bytes, $O_{i,j}$, during time interval T_j and T_{j-1}, where T_j is the time when j-th sample arrives, is calculated as follows

$$O_{i,j} = A_j\ (T_j - T_{j-1}), \tag{4}$$

where A_j is the bandwidth allocated to each flow. We assume that the target bandwidth, or weight, of each flow is equal and thus allocated bandwidth A_j is calculated as follows:

$$A_j = B / C_j, \tag{5}$$

where B and C_j are capacity of output link and the number of active flows. The definition of active flows here is the number of flows which has sent at lease one packet during a time interval.

3.2 Emulation of GPS discipline

Packet scheduling emulation defined by equations (1) to (5) is not work-conserving, i.e. residual bandwidth is not properly reused, and thus not WFQ. If there are flows whose sending rate is less than the allocated bandwidth, there have to be some residual bandwidth on the output link. To follow GPS (Generalized Processor Sharing) discipline and hence emulate WFQ scheduler, residual bandwidth has to be re-allocated to each flow according to their weights.

We use the idea of *virtual time* [16]. Virtual time elapses faster than real time when there is residual bandwidth so that more packets are transmitted from the queues and the link is fully utilized. Virtual time T^v_j, which correspond to real time T_j, is updated upon an arrival of a sampled header of flow i, as follows:

$$T^v_j = T^v_{j-1} + (T_j - T_{j-1})\ (B / \Sigma_i max(A_j, R^v_i)), \tag{6}$$

$$R^v_i = I_{i,j} / (T^v_j - T^v_{j-1}), \tag{7}$$

where R^v_i is the receiving rate of flow i in virtual time domain. Then virtual time is applied to Eq. (4) and we have:

$$O_{i,j} = A_j\ (T^v_j - T^v_{j-1}). \tag{4'}$$

These equations show that if the link is not fully utilized, residual bandwidth is allocated to each queue by accelerating the decrement of queue length. For example, if the link utilization is 50%, virtual time elapses two times faster than real time so that two times more bytes are reduced from the queue, therefore throughput of each flow doubles and the link is fully utilized.

3.3 Discarding rate calculation

Based on the queue length updated using above equations, packet discarding rate $D_{i,j}$ of flow i is calculated using random early discarding algorithm, as shown in the following equation:

$$D_{i,j} = (Q_{i,j} - Q_{min}/C_j) / (Q_{max}/C_j - Q_{min}/C_j), \tag{8}$$

where Q_{min} and Q_{max} are queue thresholds for minimum and maximum discarding rate, respectively. The thresholds are divided by the number of active flows so that the buffer capacity is appropriately shared by the flows.

3.4 Bloom filter generation

Packet discarding rate of all flows is encoded into multiple bloom filters as explained in the previous sections.

4. Simulation Results

In this section, we show some of simulation results of the WFQ emulation example. Due to lack of the space, we only show a few typical results. Variety of simulation results using web-like on-off flows, for example, will appear in the future paper.

4.1 Evaluation model

Figure 5 shows the network model used for this simulation study. The model is a dumb-bell topology and link capacity is 100 Mbps. A control server is attached to the switches and dedicated links are provided for their communication. The number of flows is 100 and each flow uses either TCP-NewReno with SACK option or constant bit rate UDP. Propagation delay between sender and receiver ranges 1msec to 126msec.

We compared three different traffic management mechanisms; (1) FIFO + tail drop, (2) FIFO + proposed WFQ emulation, and (3) per-flow queue + DRR scheduler. For (1) and (2), the switches have a single queue shared by all flows and the buffer capacity is 500KB. For (3), the switches have separate per-flow queues and the buffer capacity is either 500 KB (5 KB per queue) or 50 MB (500 KB per queue).

We set (B_{min}, B_{max}) = (500 KB, 5 MB) for TCP flows to allow more burstiness of TCP flows, and (B_{min}, B_{max}) = (50 KB, 500 KB) for UDP flows. In this experiment, we assume that there's no collisions of hash functions, thus bloom filter size is 100 bit. The control server maintains 100 bloom filters and its update for the switches is 100 times per second. We used NS2 simulator [17].

4.2 Evaluation on sampling rate

Most of control server operations including WFQ and RED emulation, as well as bloom filter generation, are performed when a sampled packet header arrives. Therefore, sampling rate is a key factor that determines server performance. Smaller sampling rate is necessary for higher server performance, but it may deteriorate link utilization, stability, and fairness among flows.

Figure 6 shows link utilization for different sampling rates. In this case, two scenarios are examined; (1) 25 TCP flows whose RTT ranges 1msec to 126msec with 5msec interval, (2) 4 sets of scenario (1) flows. This figure shows that sampling rate has to be more frequent than 1/100 to stabilize traffic control. If sampling rate is less frequent than that, arriving rate is sometimes overestimated

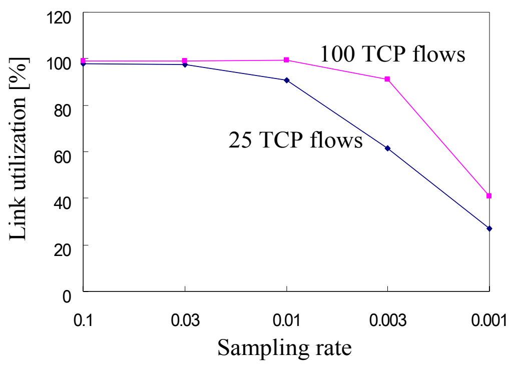

Fig. 6: evaluation of sampling rate

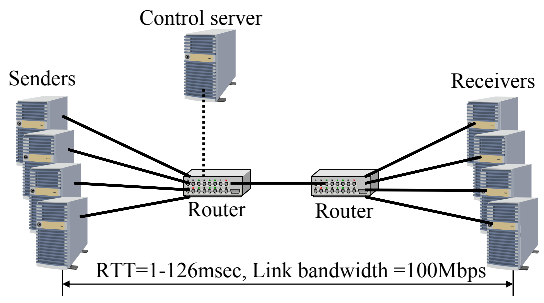

Fig. 5: evaluation model

and packet discarding rate tends to be higher, which results in throughput degradation.

4.3 Evaluation using TCP flows

Figure 7 shows average throughput of individual flows for the case where 100 TCP flows compete. In this case, fair share throughput of a flow is 1 Mbps. RTT ranges 1msec to 126msec with 5msec interval and each RTT group has 4 TCP flows. As a nature of TCP, flows with shorter RTTs tend to obtain higher throughput than longer RTT flows.

Jain's fairness index of Proposed, DRR (50MB), DRR (500KB), and Drop-tail are 0.996, 0.999, 0.365, and 0.800, respectively. This figure shows that DRR with a huge buffer achieves best fairness among flows. Highest throughput is just 1.1 times larger than lowest throughput. However, if the buffer capacity is 500KB, same as other FIFO cases, DRR shows the worst fairness because buffer capacity of each queue becomes too small. This is because the buffer capacity is strictly separated for each flow and no statistical multiplexing gain is expected. Drop-tail is somewhat better than DRR with small buffer, but the throughout difference is still large and highest throughput is 4 times larger than lowest throughput. Our mechanism achieves good fairness even with small buffer capacity. By setting B_{max} larger than the actual buffer capacity, the buffer is efficiently shared. As a result, per-flow fairness of the proposed mechanism is comparable with a DRR scheduler.

4.4 Evaluation using TCP and UDP flows

Figure 8 shows average throughput of individual flows for the case where 10 UDP flows and 90 TCP competes, thus fair share throughput is 1 Mbps. RTT of each TCP flow is fixed at 20msec. Sending rate of UDP flows ranges from 1 Mbps to 10 Mbps with 1 Mbps interval.

In this case, Jain's fairness index of Proposed, DRR (50MB), DRR (500KB), and Drop-tail are 0.998, 0.999, 0.100, and 0.317,

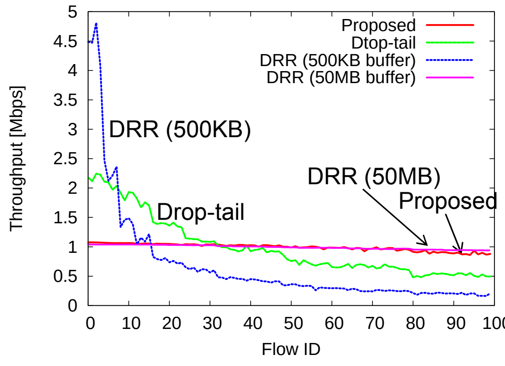

Fig. 7: throughput of TCP flows with different RTTs

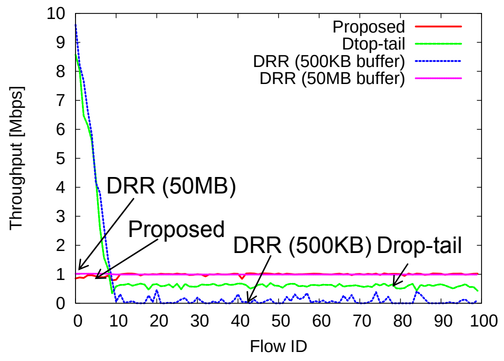

Fig. 8: throughput of co-existing UDP and TCP flows

respectively. DRR with huge buffer and our mechanism shows good per-flow fairness. With drop-tail, UDP flows obtain larger throughput and throughput of TCP flows are about half of the fair share throughput. DRR with small buffer results in very poor fairness and link utilization. Since buffer capacity allocated to each flow is very small, retransmission timeout frequently occurs and thus throughput is seriously degraded.

5. Discussions on Implementation Issues

5.1 Flow table size

Bloom filters have a possibility of false positive, and there is a tradeoff between false positive ratio and bloom filter size. False positive ratio f can be calculated by the following equation:

$$f \cong \left(1 - e^{-K\alpha\beta C/N}\right)^K$$

where α, β, and C is the ratio of large flows (i.e. flows needs to be controlled) to the entire flows, average packet discarding rate of large flows, and maximum member of concurrent active flows. For example, if we assume that number of entire flows, concurrent active flows, and large flows, is 1 million, 0.1 million, and 0.1 million, respectively, and that 10% packet discard is enough to slow down TCP flows, we obtain a false positive ratio shown in Fig. 9 for various filter sizes and number of hash functions.

If we assume that false positive ratio is 0.001, this means 0.1% of innocent flows suffer from improper packet discard. It may be possible to spread the risk of inappropriate damage by frequently changing hash functions. In this case, 0.001 false positive ratio can be understood that many flows get very small level of false packet discard. If this is acceptable, bloom filter size and flow table size become 17 Kbit with K=5. On the other hand, if we assume to have a flow table with 5-tuple, one entry size is

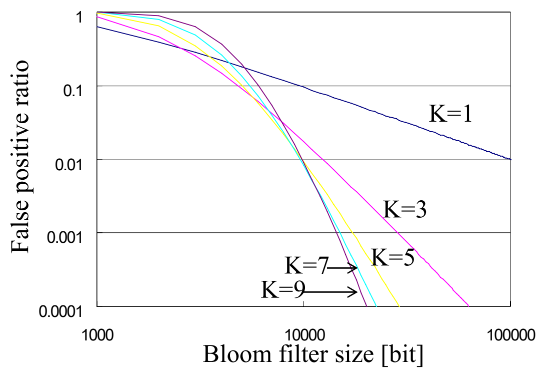

Fig. 9: false positive ratio of bloom filters

32+32+8+16+16=104 bit for an IPv4 flow. And thus a size of a flow table with 0.1 million flow entry becomes 10 Mbit. Therefore, in this particular example, flow table size is compressed 600 times smaller by using bloom filters.

5.2 Processing capability of a switch

Memory size for a flow table is small enough to be located on a corner of a logic LSI or FPGA, and there is no need to add external SRAM or CAM. Thus, access latency to the flow table can be as small as one or two clock cycles. If we assume that clock cycle of a FPGA is 100 MHz, 64 byte packets are processed in every 5 clock cycles at 10 Gbps links. Since the number of memory accesses with K=5 is 5 times per packet, we can conclude that our mechanism can be applied to switches having more than 10 Gbps links.

5.3 Processing capability of a control server

In our evaluation, we found that it is OK that sampling rate is 1/100 and bloom filter update frequency is 100 times a second. Assuming that a packet size is 1.5 KB, the number of packet headers sampled from a 10 Gbps link is 8300 per second. Therefore, the server updates packet discarding rate 8300 times a second and sends bloom filter 100 times a second, which would be small enough for modern CPU capability.

5.4 Communication bandwidth between server and switch

Assuming that header length is 40 bytes and the switch sends 8300 samples a second, the bandwidth consumed by sFlow packets is 2.7 Mbps. Also, assuming that bloom filter size is 15 Kbit and its update frequency is 100 times a second, the bandwidth consumed by the filter update is 1.5 Mbps. These communication bandwidths should be reasonably small compared to the link capacity.

6. Conclusion

In this paper, we proposed a programmable and scalable traffic management mechanism that can handle more than 10 Gbps and more than 1 million flows. The proposed mechanism consists of control server software, which enables programmable traffic management, and a hardware packet discarder for high-speed switches. We also proposed a flow table mechanism that uses a time series of bloom filters, which compresses the flow table by 1/600. We tested the proposed mechanism with a per-flow WFQ emulation and the results ware promising, although we need more evaluations of the cases where there are a large number of short-lived flows that can not be captured by sampling. More importantly, evaluation of the mechanism using more appropriate applications like regulating heavy hitters or mitigating DoS attack packets are our important future work.

7. References

[1] NSF NeTS FIND Initiative, http://www.nets-find.net/

[2] 4ward project page, http://ww.4ward-project.eu

[3] EU FP7 IST Trilogy project, http://www.trilogy-project.org

[4] GENI: Global Environment for Network Innovations, http://www.geni.net/

[5] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang, "A clean slate 4D approach to network control and management," ACM Computer Communication Review, October 2005

[6] H. Yan, D. A. Maltz, T. S. Eugene Ng, H. Gogineni, H. Zhang, Z. Cai, "Tesseract: A 4D Network Control Plane", NSDI 2007

[7] OpenFlow Switch Consortium, http://openflowswitch.org/index.php

[8] I. Stoica, S. Shenker, H. Zhang, "Core-Stateless Fair Queueing: A Scalable Architecture to Approximate Fair Bandwidth Allocations in High Speed Networks", SIGCOMM'98

[9] D. lin and R. Morris: "Dynamics of Random Early Detection", In Proc. of ACM SIGCOMM 1997

[10] R. Mahajan, S. Floyd, and D. Wetherall: "Controlling High-Bandwidth Flows at the Congested Switch", In Proc. of ICNP2001

[11] M. Shreedhar and G. Varghese: "Efficient Fair Queueing using Deficit Round Robin," IEEE/ACM Trans. On Networking, 1996

[12] A. Broder and M. Mitzenmacher, "Network Applications of Bloom Filters: A Survey", Allerton Conference 2002.

[13] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown and S. Shenker, "Ethane: Taking Control of the Enterprise", In Proc. of ACM SIGCOMM 2007

[14] PlanetLab, http://www.planet-lab.org/

[15] P. Pheal, S. Panchen, and N. McKee, "InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks," IETF, RFC3176, 1992

[16] S. J. Golestani, "A self-clocked fair queueing scheme for broadband applications", In Proc. of INFOCOM 1994

[17] Network Simulator version 2 (ns-2), available from http://www.isi.edu/nsnam/ns/