PAPER

Hierarchically Aggregated Fair Queueing (HAFQ) for Per–flow Fair Bandwidth Allocation

Ichinoshin MAKI[†], **Hideyuki SHIMONISHI**[††], **Tutomu MURASE**[††], ***and*** **Masayuki MURATA**[†], ***Members***

SUMMARY Because of the development of recent broadband access technologies, fair service among users is becoming more important goal. The most promising router mechanisms for providing fair service is per–flow traffic management. However, it is difficult to implement in high–speed core routers because per–flow state management is prohibitively expensive; thus, a large number of flows are aggregated into a small number of queues. This is not an acceptable situation because fairness degrades as the number of flows so aggregated increases. In this paper, we propose a new traffic management scheme called Hierarchically Aggregated Fair Queueing (HAFQ) to provide per–flow fair service. Our scheme can adjust flow aggregation levels according to the queue handling capability of various routers. This means the proposed scheme scales well in high–speed networks. HAFQ improves the fairness among aggregated flows by estimating the number of flows aggregated in a queue and allocating bandwidth to the queue proportionally. In addition, since HAFQ can identify flows having higher arrival rates simultaneously while estimating the number of flows, it enhances the fairness by preferentially dropping their packets. We show that our scheme can provide per–flow fair service through extensive simulation and experiments using a network processor. Since the currently available network processors (Intel IXP1200 in our case) are not high capacity, we also give extensive discussions on the applicability of our scheme to the high–speed core routers.

key words: *Fairness, Packet Scheduler, Scalability, Network Processor*

1. Introduction

Fair service among users is already one of the most important goals of those concerned with the quality of best effort traffic, and it is becoming more important as broadband access technologies such as xDSL and optical fiber remove the limits on a user's use of network resources. This means aggressive users may utilize a large amount of network resources and deteriorate quality of other users extremely [1]. Therefore, it is important to provide fair service for end users and many researches are done in order to solve this problem.

There are two main traffic management schemes for providing per–flow fair service as router mechanisms. RED [2] and SRED [3] are represented as the first main traffic management scheme. These mechanisms take an advantage of easy hardware implementation but parameter settings are very sensitive to various network factors [2]. Therefore, it is difficult to provide per–flow fair service for all users. As the second main traffic management scheme, per–flow scheduling or per–flow accounting are represented. For example, there are a lot of packet scheduling algorithms but the DRR scheme [4] should be one of the easiest to accomplish the per–flow service. When the line speed of a router is low enough that all flow states can be maintained in large capacity memories, the router can employ per–flow queueing. However, it is difficult to use per–flow queueing in high–speed core routers because large capacity memories cannot operate so fast; thus, a large number of flows are aggregated into a small number of queues. This is not a preferable situation because the more number of flows aggregated into a queue increases, the worse fairness tends to become.

In this paper, we therefore propose a new traffic management scheme called Hierarchically Aggregated Fair Queueing (HAFQ) to provide per–flow fair service. HAFQ improves the fairness among aggregated flows by estimating the number of flows aggregated in a queue and allocating bandwidth to the queue proportionally. In addition, since our scheme can identify flows having higher arrival rates simultaneously in estimating the number of active flows, it enhances the fairness by preferentially dropping their packets.

Another advantage of our scheme is that it requires no flow identification to assign a queue to a flow. The assignment can be simply implemented by hashing methods because it has only to guarantee that the difference between the number of flows aggregated into each queue is not extremely large. Flow identification is not required even in edge routers performing near per–flow queueing because our scheme allows two or more flows to occasionally be aggregated in the same queue.

We evaluate the proposed scheme through extensive simulation studies. First, we show that our scheme can estimate the number of active flows precisely in comparison to the traditional schemes. Second, we also show that the proposed scheme can provide per–flow fair service even when a large number of flows are aggregated into the same queue.

In general, a scheduling complexity can be eval-

Manuscript received March 8, 2005.
Manuscript revised May 11, 2005.
Final manuscript received 0, 2005.

[†]The author is with the Department of Information Networking, Graduate School of Information Science and Technology, Osaka University

[††]The author is with the System Platforms Research Laboratories, NEC corporation

uated if the scheduling algorithm is given. However, in today's high–speed network environment, its quantitative complexity of hardware implementation can be fully investigated neither by simulation nor by theoretical studies. We therefore implemented our scheme on Intel IXP1200 network processor [5]. Since a network processor is programmable and it can realize many router mechanisms, we can evaluate the proposed scheme in a nearly actual environment. Since the processing capacity of the IXP1200 is not high, we examine the results obtained in a relatively slow network environment and discuss the applicability in a high–speed network environment.

The remainder of the paper is structured as follows. In the next section, we propose a new scalable traffic management. In section III, we evaluate the scheme through extensive simulation studies. In section IV, we discuss the implementation design issues of the proposed scheme on the network processor and evaluate its scheduling complexity through experimental measurements. Finally, we conclude in section V.

2. Hierarchically Aggregated Fair Queueing (HAFQ)

2.1 Outline

The basic mechanism of our scheme is illustrated in Fig. 1. When a packet arrives at the router, a 16-bit CRC hashing function assigns it to a queue. It is because it can perform good load balancing [6]. Then, the number of active (long–lived) flows in each queue is estimated. We are interested in fairness among long–lived flows. Therefore, we do not consider short–lived flows in this paper. The number of flows is estimated by using a zombie list [3], which is a short history of newly arrived flows and is prepared for each queue. This means that the number of flows is estimated without maintaining the states of all active flows. And because the zombie list also helps identifying flows whose having high packet arrival rates, fairness among aggregated flows in the same queue can be improved by dropping those packets preferentially.

As output operations, our scheme allocates bandwidth to the queue according to the number of flows aggregated into each queue. Then, our scheme forwards a packet from the queues using the DRR scheduling.

2.2 Zombie List

A zombie list is a table of constant size in which is a short history of newly arrived flows. Each row in this table contains a *flow ID* and a *packet counter*, and the list is revised every time a packet arrives at the router. An entry in the zombie list is called *zombie*. When a packet arrives at the router, it performs as below.

- Search into a zombie list.

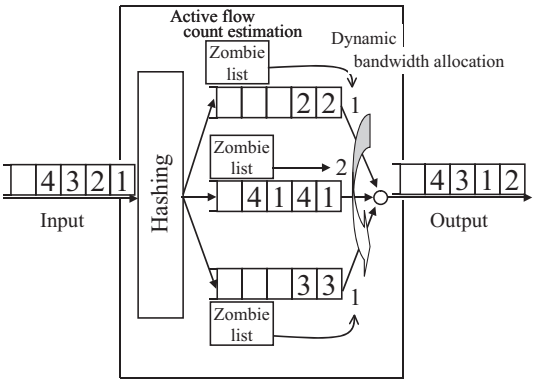


Fig. 1 Outline of the proposed scheme.

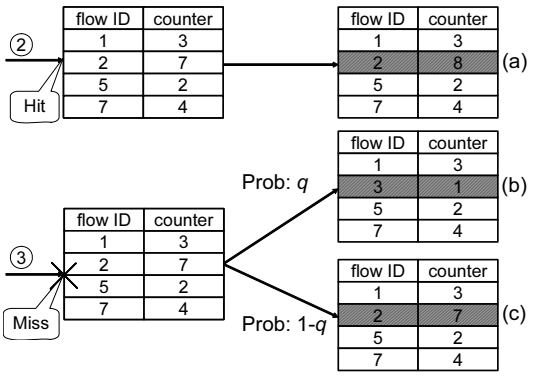


Fig. 2 Zombie list.

- When the flow ID of the packet matches a flow ID in the list, the corresponding packet counter is increased by one. This is called *Hit*.
- When no entry matches, a row is selected randomly.
  - * With probability q, the flow ID of the new packet is written into that row and the corresponding packet counter is set to 1. This is called *Swap*.
  - * Otherwise (i.e., with probability $1 - q$), nothing is done. This is called *No-swap*.

Fig. 2 shows an example in which the flow ID of an arriving packet matches the second entry in the zombie list and the corresponding packet counter is incremented by 1 (Fig. 2(a)). It also shows examples of what happens when the new flow ID does not match any of the entries in the zombie list. With probability q, the flow ID is written into a randomly selected entry and the corresponding packet counter is reinitialized (Fig. 2(b)). With probability $1-q$, the zombie list is not changed (Fig. 2(c)).

2.3 Estimating the Number of Flows

In [3], a scheme which estimates the number of flows aggregated in a queue is proposed but the number of

active flows is estimated almost accurately only when the arrival rates of all flows are equal. Otherwise, the estimated number is too small.

We therefore propose a more accurate estimation scheme that works appropriately even when the arrival rates of flows differ, which is common in an actual situation. In the proposed scheme, the arrival rates of incoming flows are estimated and their average is calculated. The number of flows can be derived from the average rate because there is the following relation between the average arrival rate λ_{avg} and the number of flows N.

$$\begin{aligned} \lambda_{avg} &= \frac{\sum_{i=1}^{N} \lambda_i}{N} \\ N &= \frac{\sum_{i=1}^{N} \lambda_i}{\lambda_{avg}} \end{aligned} \tag{1}$$

where λ_i is the arrival rate of flow i. The above equations indicate that the number of flows can be estimated by dividing the total arrival rate by the average arrival rate. Note that these equations hold when the arrival rates of the flows differ.

Now we define R_i as the ratio of the arrival rate of flow i to the total arrival rate for the same queue, i.e.,

$$R_i = \frac{\lambda_i}{\sum_{i=1}^{N} \lambda_i} \tag{2}$$

In the following, we will estimate the number of flows by deriving R_i using a zombie list. Here we assume that the packet length is fixed, but the scheme is easily extended to handle variable packet lengths.

Assume that a packet of flow i arrives at queue k and that zombie list k is updated. Let M denote the number of entries in a zombie list. If entry j ($1 \leq j \leq M$) is replaced by a newly arrived flow, the arrival rate of the flow that had been registered in that entry is estimated by using the packet counter value of the entry before the entry is replaced. This is because the maximum value of the packet counter is proportional to the rate of the flow.

When we define P_1 as the probability that a flow in a entry is replaced before packets of the flow arrives again (i.e., the probability that the maximum value of the packet counter is 1), P_1 is given by

$$\begin{aligned} P_1 &= (1-R_i)a + (1-R_i)(1-a)(1-R_i)a \\ &\quad + \{(1-R_i)(1-a)\}^2(1-R_i)a \\ &\quad + \cdots + \{(1-R_i)(1-a)\}^n(1-R_i)a \\ &= \frac{(1-\sum_{j=1}^{M} R_{X_j})\frac{q}{M}}{(1-\sum_{j=1}^{M} R_{X_j})\frac{q}{M} + R_i} \end{aligned}$$

where X_j denotes the flow ID registered in entry j and a denotes the probability that an entry is replaced by a newly arrived flow under the condition that an arrived packet matches no entry. Namely,

$$a = \frac{1-\sum_{j=1}^{M} R_{X_j}}{1-R_i} \times \frac{q}{M}$$

In the same way, the probability P_n that the packet counter is increased to n before the entry is replaced. P_n is given by

$$\begin{aligned} P_n &= R_i^{n-1}P_1 + (1-R_i)(1-a)R_i^{n-1}P_1 \\ &\quad + \{(1-R_i)(1-a)\}^2 R_i^{n-1}P_1 \\ &\quad + \cdots + \{(1-R_i)(1-a)\}^n R_i^{n-1}P_1 \\ &= \frac{R_i^{n-1}(1-\sum_{j=1}^{M} R_{X_j})\frac{q}{M}}{\{(1-\sum_{j=1}^{M} R_{X_j})\frac{q}{M} + R_i\}^n} \end{aligned}$$

Therefore, the expectation E_i for the maximum value of the packet counter is given by

$$E_i = \sum_{i=1}^{\infty} i\, P_i = \frac{R_i}{(1-\sum_{j=1}^{M} R_{X_j})\frac{q}{M}} + 1 \tag{3}$$

Now let R_i be unknown and let $\widetilde{R}_i$ denote the estimation for R_i. If we assume that the packet counter value reaches $\widetilde{E}_i$ before the entry is replaced, $\widetilde{R}_i$ can be derived using Eq. (3) as follows:

$$\widetilde{R}_i = \left(1 - \sum_{j=1}^{M} R_{X_j}\right) \frac{q}{M}(\widetilde{E}_i - 1)$$

If we assume that no entries in the zombie list are swapped, the probability p that incoming packets match one of the entries (i.e., the probability of a Hit) approaches the sum of the rates of the flows in the zombie list: $\sum_{j=1}^{M} R_{X_j}$. Therefore, if we choose the smaller value for the swapping probability q, the sum of R_{X_j} can be approximated by the probability p. Thus, $\widetilde{R}_i$ can be derived by the following equation:

$$\widetilde{R}_i = (1-p)\frac{q}{M}(\widetilde{E}_i - 1) \tag{4}$$

Then, the scheme computes the average of $\widetilde{R}_i$. Since a flow having a higher arrival rate is counted to the average arriving rate more frequently than other flows, the average is overestimated if some of the flows have higher arrival rates. Since flow i is registered in the zombie list $\widetilde{R}_i/\widetilde{E}_i$ times per unit time, $\widetilde{R}_i$ should be counted into the average with the weight $(\widetilde{R}_i/\widetilde{E}_i)^{-1}$. Therefore, the average $\widetilde{R}_{avg}$ is given by

$$\begin{aligned} \widetilde{R}_{avg} &= \left\{1 - \beta\left(\frac{\widetilde{E}_i}{\widetilde{R}_i}\right)\right\}\widetilde{R}'_{avg} + \beta\left(\frac{\widetilde{E}_i}{\widetilde{R}_i}\right)\widetilde{R}_i \\ &= \left\{1 - \frac{\alpha}{1-p}\cdot\frac{\widetilde{E}_i}{\widetilde{E}_i - 1}\right\}\widetilde{R}'_{avg} + \beta\widetilde{E}_i \end{aligned}$$

where α is a predetermined value which is $\frac{\beta M}{q}$ and β is a smoothing parameter for the average. Finally, the estimated number of flows accommodated in the queue

is calculated by $1/\widetilde{R}_{avg}$ using Eqs. (1) and (2).

$$N = \frac{1}{\widetilde{R}_{avg}} \tag{5}$$

If the number of flows is no more than the number of entries in a zombie list and all incoming packets are matched with one of the entries, the packet counter can increase infinitely. Therefore, we introduced another mechanism to deal with this problem but do not describe it in this paper because of the space limitation.

2.4 Preferential Packet Dropping Using Packet Counters

Our scheme improves fairness among flows aggregated in the same queue by detecting the flows having higher arrival rates and preferentially dropping the packets of these flows. Since Eq. (4) shows that the packet counter value is proportional to the packet arrival rate, the packets of flows having higher arrival rates can be detected easily. The proposed scheme therefore drops the incoming packet if the packet counter value is more than the average of the packet counter value and the queue length is greater than half of the buffer capacity.

We incorporate a shared buffer; this means that the maximum size of each queue is not fixed. Therefore, there is no serious performance degradation even when the number of flows accommodated in each queue considerably differs. If the maximum size of each queue is fixed in such a case, our scheme may be as good as the tail drop for buffer control. In this paper, the size of shared buffer equals to the value of bandwidth–delay product.

3. Simulation Results

In simulation, we used the single–bottleneck network topology shown in Fig. 3. We assumed that the bandwidth of the access links and the bottleneck link is 155 Mbps, and the propagation delays of these links are respectively 0.1 and 1 ms. All hosts use TCP or UDP (3.2 Mbps) and they have an infinite amount of data to transmit. The number of entries in one zombie list is four. All simulations were run using the NS simulator [7].

3.1 Estimated Number of Flows

We evaluated the flow number estimation of our scheme and compared it with the estimation of SRED. Figures 4(a)–(c) show the estimated number of flows aggregated in a queue and the number of active flows. In these figures, "HAFQ w/o DROP" denotes our scheme without the preferential packet–dropping using packet counters and "HAFQ" denotes our scheme with the packet–dropping. We assumed that one flow starts to transmit at time 0, that the number of flows doubles every 2 seconds until it reaches 64 and that all these flows are aggregated in one queue.

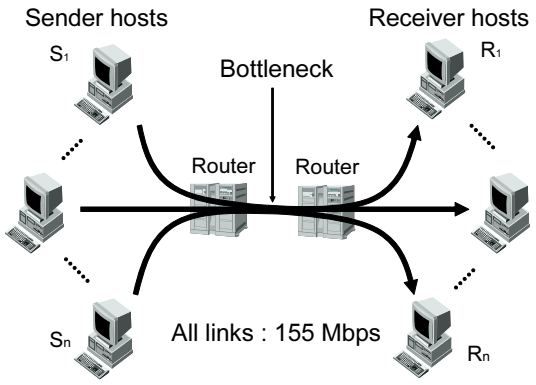
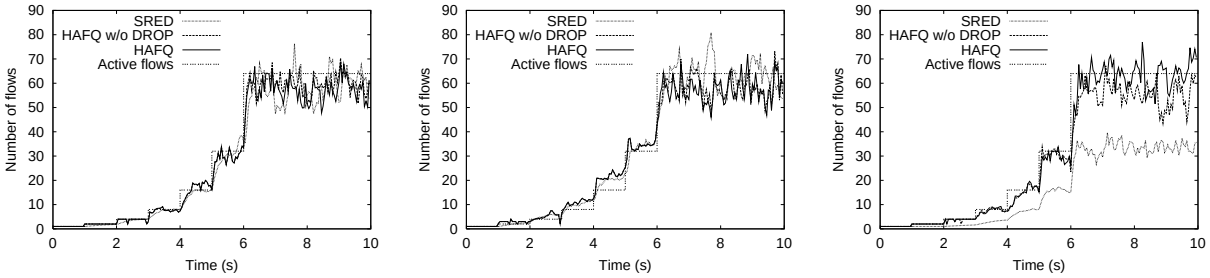


Fig. 3 Single-link model.

In Fig. 4(a), all flows are TCP flows and their RTTs are same. In this case, all three schemes give approximately correct numbers. In Fig. 4(b), half of the access links have 1 ms propagation delays and the other half have 0.1 ms propagation delays. This figure shows that RTTs have little influence on the estimated number of flows. In Fig. 4(c), half the flows are TCP flows and the other half are UDP flows. In this case, the number of flows estimated by SRED are much smaller than the correct values. Since the arrival rates of UDP flows are much higher than those of TCP flows, SRED counts flows as if the only flows in the network are UDP flows. In our scheme without the packet–dropping, however, the estimated numbers of flows are only slightly less than the correct values. With the packet–dropping, the estimation is improved and the error is less than 15%. This is because packets of UDP flows are preferentially dropped and the arrival rates of all flows become more uniform.

According to [8], the number of active flows in backbone networks reaches about several tens of thousands. However, we think we do not need to run simulations in such a case because the number of flows accommodated to a queue is expected to be small. As shown in Subsection 4.3.4, our scheme would be able to have 1K queues in backbone routers. This means that the number of flows accommodated to a queue would be about several tens even in backbone routers. We therefore think our simulation is very reasonable to evaluate the flow number estimation. In addition, the estimated number of flows has relation to the average arrival rate of flows accommodated to a queue as shown in Eq. 5. Therefore, our scheme shows good estimation independently of the number of active flows. In Subsection 3.2 and 3.3, we run simulation when the number of active flows is small as well as in this subsection. This reason is shown as described above.

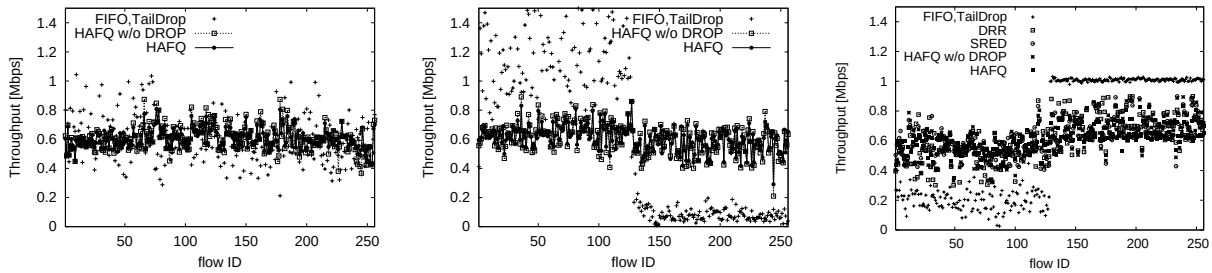


(a) TCP flows in a homogeneous environment

(b) TCP flows in a heterogeneous environment

(c) TCP and UDP flows in a homogeneous environment

Fig. 4 The estimated number of flows in SRED and the proposed scheme.



(a) TCP flows in a homogeneous environment

(b) TCP flows in a heterogeneous environment

(c) TCP and UDP flows in a homogeneous environment

Fig. 5 Throughput of TCP and UDP flows.

3.2 Throughput of Each Flow on the Middle Aggregation Level

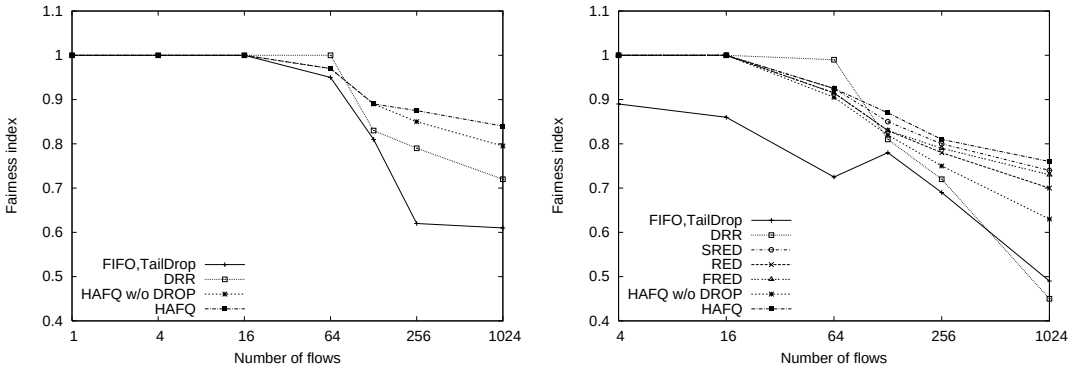
We next evaluated our scheme using 64 queues and determining the ones that flows were stored in by hashing the flow IDs, in comparison with the FIFO scheme using tail drop for buffer control. In this evaluation, the number of flows was 256. We run each simulation during 10 sec.

Fig. 5(a) shows the individual throughput of all TCP flows. This figure shows that the throughput of the FIFO scheme differs and the proposed scheme improves fairness among flows. In Fig. 5(b), half of the TCP flows have longer RTTs as evaluated in Fig. 4(b). The FIFO scheme gives TCP flows with shorter RTTs more bandwidth than those with longer RTTs, and our scheme decreases the difference between flows with different RTTs.

In Fig. 5(c), half of the flows are TCP flows and the other half are UDP flows (1.2 Mbps). In this simulation, our scheme was also compared with DRR and SRED scheme. In the DRR scheme, each flow is accommodated in its own queue if the number of active flows is less than 64; otherwise the extra flows are accommodated in one of the queues by hashing the flow IDs. Since UDP flows do not have congestion control mechanism, we consider serious unfairness between TCP and UDP flows. We therefore evaluate the effectiveness of our scheme comparison to traditional schemes. The FIFO scheme gives UDP flows much more bandwidth than TCP flows and our scheme improves fairness between TCP and UDP flows. DRR and SRED scheme improve fairness between TCP and UDP flows but they cannot achieve good fairness comparison to our scheme with the packet–dropping. This is because, in the DRR scheme, there is a difference in the number of flows aggregated in one queue, while the same amount of the bandwidth is allocated to each queue, and the SRED scheme cannot estimate the number of active flows correctly. In these three cases, the packet–dropping of our new scheme further improves fairness among flows.

3.3 Fairness Index on the High Aggregation Level

The throughput of a large number of flows was next examined in order to evaluate a fairness property of our scheme in a large–scale–network environment. In this evaluation, our scheme was compared with the FIFO scheme and with a DRR per–flow scheduling scheme. Based on a core router environment, we suppose that both our new scheme and the DRR scheme have 64



(a) Only TCP flows (b) TCP and UDP flows

Fig. 6 Fairness index versus the number of flows.

queues. Here we use a Fairness Index as the fairness measure. Its value f is near 1 if the throughputs of all flows are equal, and it gets smaller as differences in throughput increase. It is calculated as follows

$$f(x_1, x_2, x_3, \cdots, x_N) = \frac{(\sum_{i=1}^{N} x_i)^2}{N \sum_{i=1}^{N} {x_i}^2} \tag{6}$$

where x_i is the throughput of flow i, and N be the total number of flows.

Figures 6(a) and 6(b) show the fairness index plotted against the number of flows. In Fig. 6(a) all flows are TCP flows, and in Fig. 6(b) half of the flows are UDP flows. In Fig. 6(b), our scheme was also compared with the modified versions of RED, SRED, and FRED [9] scheme. For example, the RED scheme is the same as the modified version of "HAFQ w/o drop" with buffer control of RED scheme. The fairness of FIFO scheme becomes worst as the number of flows increases and, compared to the other schemes, its fairness is worst in any situations. The DRR scheme provides excellent fairness if the number of flows is less than the number of queues (i.e., 64), but it becomes increasingly less fair as the number of flows increases beyond 64. This is because there is a difference in the number of flows aggregated in one queue, while the same amount of the bandwidth is allocated to each queue. On the other hand, since our scheme dynamically allocates bandwidth in proportion to the estimated number of flows, its fairness index decreases only gradually as the number of flows increases. RED, SRED, and FRED scheme show less fair than HAFQ scheme. In these schemes, parameter settings are very sensitive to various network factors. Therefore, it is difficult to provide good fairness for all flows.

When the number of flows is 64, the fairness index of our scheme is worse than that of the DRR scheme. This is because exactly one flow is accommodated in one queue in the DRR scheme, whereas flows are accommodated in the queues randomly in our scheme. For most numbers of flows, however, our scheme provides better fairness and its fairness is less sensitive to the number of flows. These figures show that the packet–dropping is especially effective when there are many ill–behaved flows in the network.

4. Implementation Design Issues of HAFQ on the IXP1200 Network Processor

In this section, we discuss the implementation design issues of HAFQ on the IXP1200 network processor [5], which has six microengines for packet forwarding operations, each of which can deal with four threads (contexts) concurrently. See also Fig. 7. A microengine is a 32–bit RISC programmable data engine and a thread can realize multiple control streams in one program. In addition, it provides (1) an SDRAM unit to access low cost, high bandwidth memory for mass data, (2) an SRAM unit for very high bandwidth memory to store lookup tables and other data for packet processing, and (3) a scratch pad memory which is an embedded memory unit.

4.1 Implementation Outline

The microengines #1 through #4 perform packet input operations including packet header verification, destination address lookup, header modification of IPv4 packets and HAFQ ingress operations of IPv4 packets. Then, the microengines #5 and #6 perform packet output operations including determination of the transmitting queue by the DRR scheduling and dynamic bandwidth allocation. This "2–to–1 allocation" is based on the suggestion described in [10]. We use the SDRAM unit to deploy packet data as a shared buffer pool. Since it has larger memory capacity and higher memory bandwidth than the SRAM unit, it is suitable to the shared packet buffer pool. On the other hand, the SRAM unit holds the routing table, zombie list, and

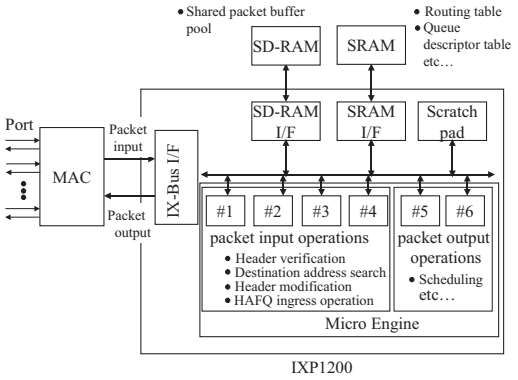


Fig. 7 Our task assignment on a network processor.

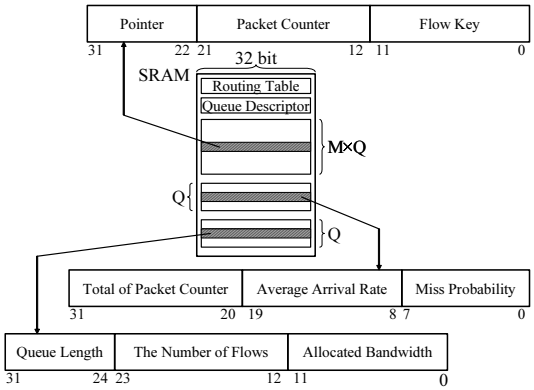


Fig. 8 Memory map on the SRAM unit.

other states which are needed in performing HAFQ operations.

4.2 Memory Model

A memory capacity is severely limited in high–speed routers. Thus, we carefully designed the memory model of HAFQ. Fig. 8 shows the memory model for HAFQ operations. The bit allocations for each variable are not explained in this paper because of the space limitation but they are based on the assumption that the number of active flows in each line interface reaches several tens of thousands, and at least a few dozens of queues can be maintained in the router. We assume that parameters $\{q,\ M\}$ are $\{0.01,\ 4\}$ to determine the bit allocations for packet counters.

Based on these bit allocations, the required memory capacity for the zombie lists is determined by $M \times Q \times 32$ bit, where Q and M represent the number of queues and the number of entries in a zombie list, respectively. Both of the required memory capacity for the flow count estimation and packet scheduling are $Q \times 32$ bit. Therefore, the total required memory capacity for implementing HAFQ is

$$M \times Q \times 32 + Q \times 32 + Q \times 32 = 32 \cdot Q \cdot (2+M)\ \ [bit] \quad (7)$$

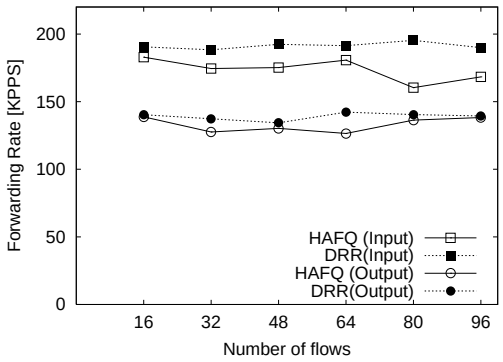


Fig. 9 Packet forwarding rate.

The memory capacity that edge routers can have in the case of using an off–chip memory is up to 4 Mbytes in the current memory technologies. Since our scheme can have about 512K queues, it would be possible to provide per–flow scheduling. On the other hand, core routers use an on–chip memory because the line speed is very high in backbone networks. Since its memory capacity is about 32 Kbytes, our scheme can have 1K queues from Eq. (7).

4.3 Implementation Evaluation

We use the IXP1200 Developer Workbench [11] for simulation. We suppose the network model shown in Fig. 3. 96 sender hosts and 96 receiver hosts are connected through the router, and each sender host generates IP packets whose length is a fixed size of 512 byte. Each sender host generates packet at 100 Mbps (constant bit rate), and those sender hosts have an infinite amount of data to transmit. We note that our current experimental implementation does not perform packet header modification or routing table look–up, because we intend to evaluate the overhead of the proposed scheme.

4.3.1 Evaluation on Packet Processing Capacity

In this subsection, we investigate the packet processing capacity of the proposed scheme by comparing to that of the DRR scheme. These two schemes have 16 queues, and additionally HAFQ has two entries in each zombie list.

Fig. 9 shows a packet processing capability of packet input operations and output operations. In packet input operations, the processing capability of HAFQ is about 5 percent lower than that of DRR. This is because HAFQ operations require additional instructions such as searching the zombie list and estimating the number of flows. As for packet output operations, although the processing capability of HAFQ is lower than that of DRR, the performance degradation is limited.

4.3.2 Fairness Comparison for Different Number of Flows

In this subsection, we evaluate a fairness property of our scheme and the DRR scheme as the number of flows increases. In this evaluation, we use the Fairness Index as the fairness measure (See Eq. (6)). Both our scheme and the DRR scheme have 16 queues, and the number of entries in each zombie list is two in our scheme. In the DRR scheme, each flow is accommodated in its own queue if the number of active flows is less than 16; otherwise, the extra flows are accommodated in one of the queues randomly. On the other hand, our scheme determines the accommodated queue randomly, even if the number of flows is less than 16.

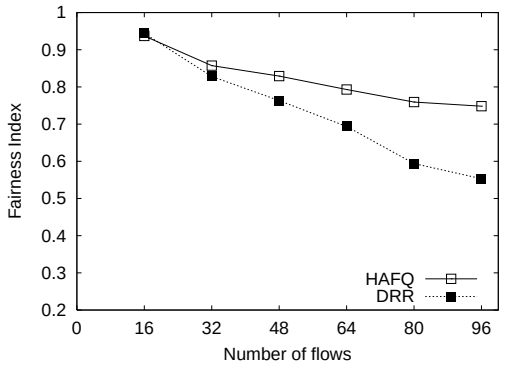


Fig. 10 Fairness in the case of the different number of flows.

Fig. 10 shows the fairness index against the number of active flows. The same tendency can be observed as in the simulation experiments presented in Subsection 3.3. We can confirm that the DRR scheme provides excellent fairness if the number of flows is equal to the number of queues, but it provides degraded fairness as the number of flows increases. In our scheme, on the other hand, although the fairness index decreases gradually as the number of flows increases, its high fairness can be provided.

4.3.3 Fairness Comparison for Memory Requirements

We evaluate the effect of required memory capacity, i.e., implementation cost, on the fairness property. Fig. 11 shows the fairness index against the required memory capacity. The number of active flows is 96. In the DRR scheme, the number of queue is changed from 32 to 96; thus, its memory requirement ranges from 128 byte to 384 byte. Now, recalling that 4 byte is necessary for managing one queue. In the HAFQ scheme labeled by "HAFQ / queue", the number of entries is fixed at two in each zombie list and the number of queues is changed as 8, 16, and 24; thus, required memory capacity is 128, 256, and 384 bytes. Also, in the HAFQ scheme labeled by "HAFQ / zombie", the number of entries in a zombie list is changed from 2, 3, $\cdots$, 10 while the number of queues is fixed at eight; thus the memory requirement ranges from 128 byte to 384 byte.

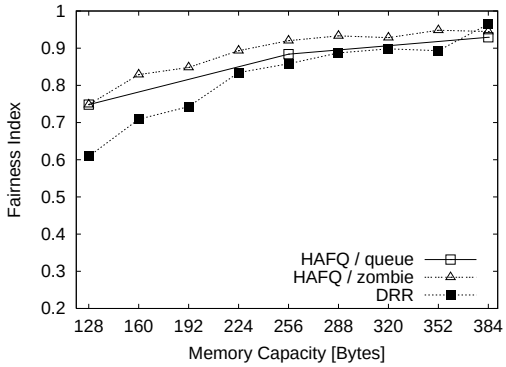


Fig. 11 Fairness in the case of the different memory capacity.

Fig. 11 shows that the fairness of the HAFQ scheme is better than that of the DRR scheme and HAFQ requires only one third of the memory capacity compared to the DRR scheme to achieve the same fairness level. The fairness index is improved as the available memory capacity becomes large. This is because the number of flows aggregated in one queue is small when the number of queues is large, therefore, accuracies for both flow count estimation and preferential packet dropping are improved.

In the two HAFQ schemes, HAFQ/zombie provides better fairness than HAFQ/queue. This means that the memory capacity should be used for the larger zombie lists rather than increasing the queue number. However, the computational cost for the search increases as the zombie list becomes large; thus, there would be specific limits for the zombie list size in specific environments. This is a design choice for the trade–off between memory capacity and processing performance.

In general, there is a difference in the memory capacity between edge routers and core routers. Therefore, a scalable packet scheduling scheme must be able to minimize the performance degradation as the memory capacity becomes small. Our scheme provides more excellent fairness than the DRR scheme as the memory capacity becomes limited. In other words, our scheme provides a more scalable packet scheduling than the DRR scheme.

4.3.4 Applicability to High–Speed Routers

The processing capacity of the IXP1200 is not high, and packet processing capacity is limited up to about 200 KPPS in our experiments. However, the router with a 10 Gbps line interface should have 100 times larger packet processing capacity than that of IXP1200.

Therefore, we last discuss the capability of our scheme for 10 Gbps line interfaces.

Whether packet scheduling schemes can accommodate 10 Gbps line interfaces or not greatly depends on the processing capacity and memory access bandwidth of routers. With such high–speed lines, routers can spend only 40 ns in processing one packet, and complex operations would easily lead to the performance degradation. However, the processing capacity has been rapidly improved and this problem can be expected to be solved in the near future. On the other hand, the memory access bandwidth seems to be an obstacle even in the future. Our scheme requires 13 memory accesses and the DRR scheme requires 7 memory accesses in packet input / output operations. Therefore, they require 325M and 175M memory accesses in 10 Gbps line interfaces. Thus, those schemes must not use an off–chip memory but an on–chip memory. If we use the on–chip memory, the memory access bandwidth is not a problem, but the memory capacity is very limited. However, even in such a circumstance, our scheme can accommodate many active flows in the practical number of flows keeping fairness on some level in case of the limited memory usage; for example, if our scheme has six entries in each zombie list and 1K queues, the required memory capacity is 32 Kbytes as can be estimated from Eq. (7). That is, our scheme can use an on–chip memory. On the other hand, the DRR scheme requires 6 Mbyte memory capacity for 192,000 queues for achieving the same fairness index.

5. Conclusion

The new scalable queue management scheme described in this paper provides fair per–flow service in backbone networks. The scheme estimates the number of flows aggregated in a queue and allocates the bandwidth to the queue proportionally. It also improves fairness among flows in the same queue by preferentially discarding the packets of flows having higher arrival rates. We have shown the effectiveness of our scheme through extensive simulation and experiments.

For future works, we will evaluate the performance of the HAFQ algorithm in the other situations. We also evaluate its performance by actual experiments.

References

[1] R. Mahajan and S. Floyd, "Controlling high bandwidth flows at the congested router," ACM International Conference on Nework Protocols (ICNP), Nov. 2001.
[2] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," IEEE/ACM Transactions on Networking, vol.1, no.4, pp.397–413, Aug. 1993.
[3] T.J. Ott, T. Lakshman, and L. Wong, "SRED: Stabilized RED," Proceedings of IEEE INFOCOM 1999, pp.1346–1355, March 1999.
[4] M. Shreedhar and G. Varghese, "Efficient fair queueing using deficit round robin," IEEE/ACM Transactions on Networking, vol.4, no.3, pp.375–385, June 1996.
[5] "Intel IXP1200." available at `http://developer.intel.com/design/network/products/npfamily/ixp1200.htm`.
[6] Z. Cao, Z. Wang, and E. Zegura, "Performance of hashing–based schemes for Internet load balancing," Proceedings of IEEE INFOCOM 2000, pp.332–341, March 2000.
[7] "UCB/LBNL/VINT network simulator - ns (version 2)." available at `http://www-mash.cs.berkeley.edu/ns/`.
[8] K. Claffy, H.W. Braun, and G. Polyzos, "A parameterizable methodology for Internet traffic flow profiling," IEEE Journals on Selected Areas in Communication, vol.13, no.8, pp.1481–1494, March 1995.
[9] D. Lin and R. Moriss, "Dynamics of random early detection," Proceedings of ACM SIGCOMM'97, pp.127–137, Sept. 1997.
[10] T. Spalink, S. Karlin, and L. Peterson, "Evaluating network processors in IP forwarding," tech. rep., Technical Report TR-626-00, Department of Computer Science, Princeton University, Nov. 2000.
[11] "IXP1200 Developer Workbench." available at `http://www.intel.com/design/network/products/npfamily/sdk2.htm`.

Ichinoshin Maki Ichinoshin Maki received his M.E. degree from Graduate School of Engineering Science, Osaka University in 2002. Since April 2002, he has been currently an assistant professor of Graduate School of Information Science and Technology, Osaka University. His research interests are in the area of traffic management in high–speed networks. He is a member of IEICE.

Hideyuki Shimonishi Hideyuki Shimonishi received his M.E. and PhD degree from Graduate School of Engineering Science, Osaka University, Osaka, Japan, in 1996 and 2002, respectively. He joined NEC Corporation in 1996 and has been engaged in research on traffic management in high-speed networks, switch and router architectures including cell/packet scheduling algorithms and buffer management mechanisms, and traffic control protocols. He was a visiting scholar at Computer Science Department, University of California at Los Angeles, to study next generation transport protocols. He is a member of ACM and IEICE.

Tutomu Murase Tutomu Murase was born in Kyoto, Japan in 1961. He received his M.E. degree from Graduate School of Engineering Science, Osaka University, Japan, in 1986. He also received his PhD degree from Graduate School of Information Science and Technology, Osaka University in 2004. He joined NEC Corporation in 1986 and has been engaged in research on traffic management for high-quality and high-speed internet. His current interests include TCP session layer traffic control, network traffic traceability and network security. He is a member of IEICE. He was a secretary and has been a member of steering committee of Communication Quality Technical Group in IEICE. He is also a member of steering committee of Information Network Technical Group in IEICE. He is a vice chair person of Next Generation Network working group in 163rd Committee on Internet Technology (ITRC).

Masayuki Murata Masayuki Murata received the M.E. and D.E. degrees in Information and Computer Sciences from Osaka University, Japan, in 1984 and 1988, respectively. In April, 1984, he joined Tokyo Research Laboratory, IBM Japan, as a Researcher. From 14 September 1987 to January 1989, he was an Assistant Professor with Computation Center, Osaka University. In February 1989, he moved to the Department of Information and Computer Sciences, Faculty of Engineering Science, Osaka University. From 1992 to 1999, he was an Associate Professor in the Graduate School of Engineering Science, Osaka University, and from April 1999, he has been a Professor of Osaka University. He moved to Advanced Networked Environment Division, Cybermedia Center, Osaka University in April 2000. He has more than two hundred papers of international and domestic journals and conferences. His research interests include computer communication networks, performance modeling and evaluation. He is a member of IEEE, ACM, The Internet Society, IEICE and IPSJ.

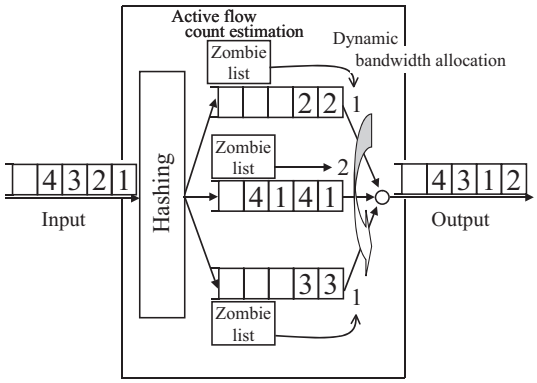


Fig. 1 Outline of the proposed scheme.

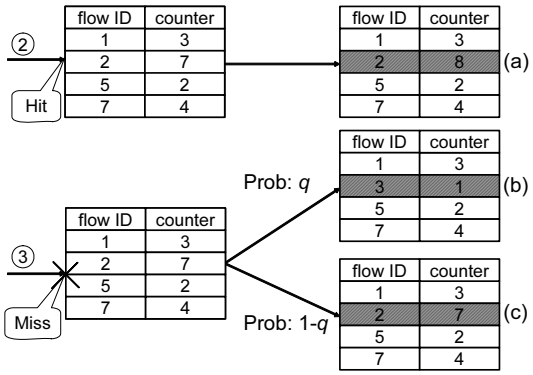


Fig. 2 Zombie list.

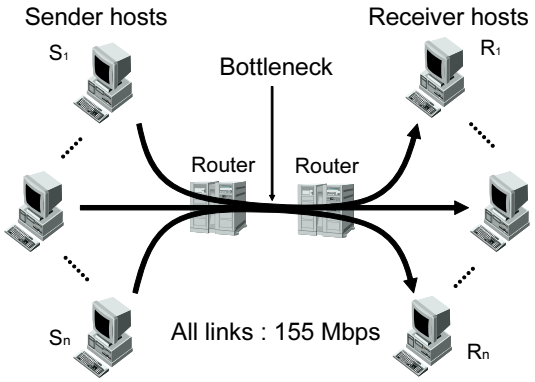
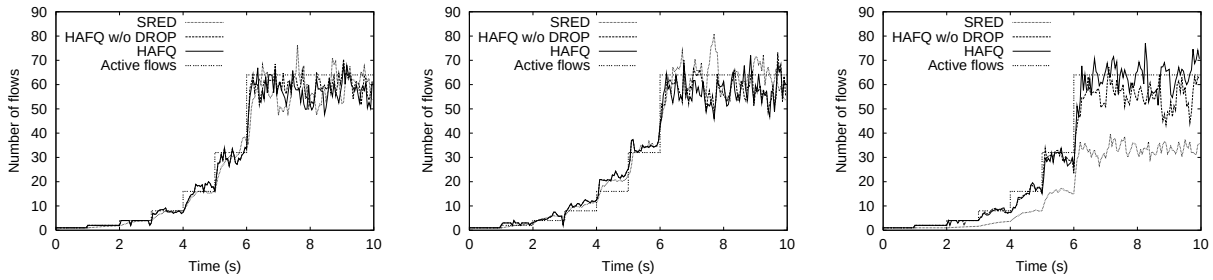


Fig. 3 Single-link model.

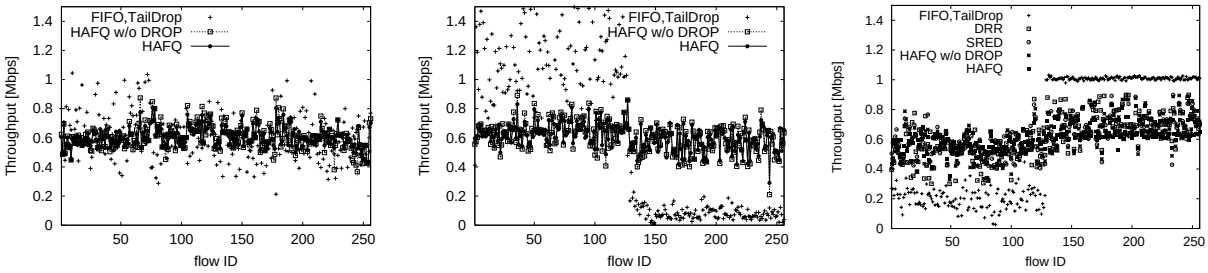


(a) TCP flows in a homogeneous environment

(b) TCP flows in a heterogeneous environment

(c) TCP and UDP flows in a homogeneous environment

Fig. 4 The estimated number of flows in SRED and the proposed scheme.

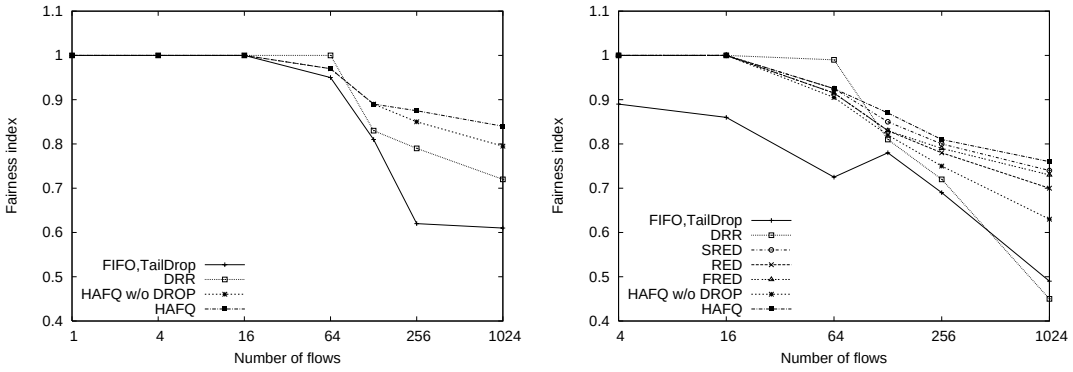


(a) TCP flows in a homogeneous environment

(b) TCP flows in a heterogeneous environment

(c) TCP and UDP flows in a homogeneous environment

Fig. 5 Throughput of TCP and UDP flows.



(a) Only TCP flows (b) TCP and UDP flows

Fig. 6 Fairness index versus the number of flows.

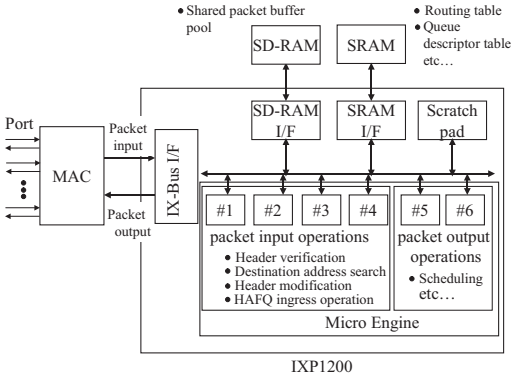


Fig. 7 Our task assignment on a network processor.

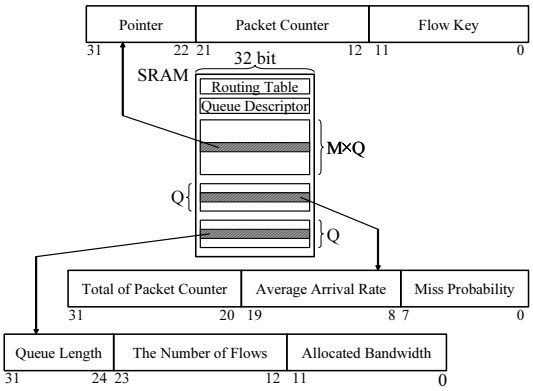


Fig. 8 Memory map on the SRAM unit.

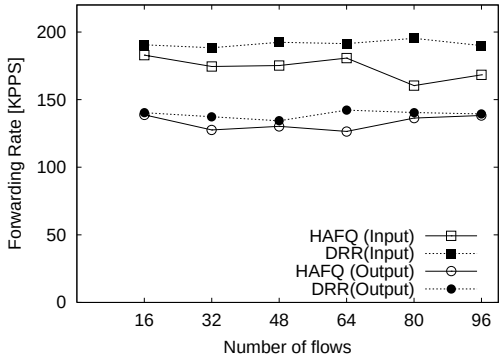


Fig. 9 Packet forwarding rate.

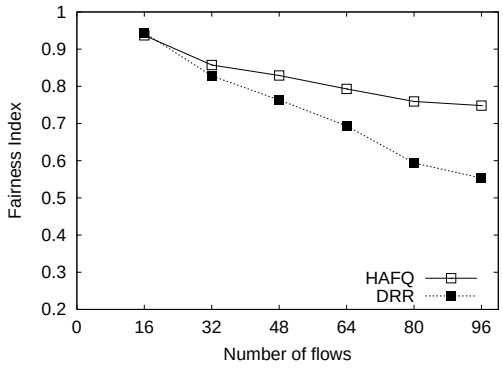


Fig. 10 Fairness in the case of the different number of flows.

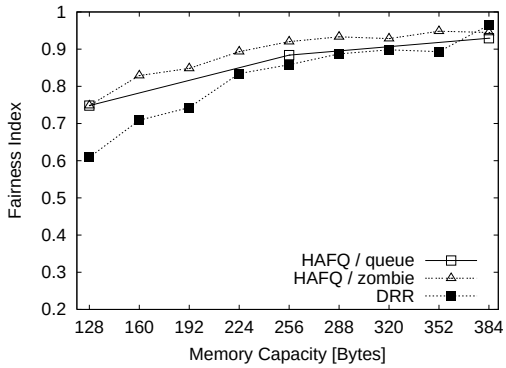


Fig. 11 Fairness in the case of the different memory capacity.

Ichinoshin Maki Ichinoshin Maki received his M.E. degree from Graduate School of Engineering Science, Osaka University in 2002. Since April 2002, he has been currently an assistant professor of Graduate School of Information Science and Technology, Osaka University. His research interests are in the area of traffic management in high–speed networks. He is a member of IEICE.

Hideyuki Shimonishi Hideyuki Shimonishi received his M.E. and PhD degree from Graduate School of Engineering Science, Osaka University, Osaka, Japan, in 1996 and 2002, respectively. He joined NEC Corporation in 1996 and has been engaged in research on traffic management in high-speed networks, switch and router architectures including cell/packet scheduling algorithms and buffer management mechanisms, and traffic control protocols. He was a visiting scholar at Computer Science Department, University of California at Los Angeles, to study next generation transport protocols. He is a member of ACM and IEICE.

Tutomu Murase Tutomu Murase was born in Kyoto, Japan in 1961. He received his M.E. degree from Graduate School of Engineering Science, Osaka University, Japan, in 1986. He also received his PhD degree from Graduate School of Information Science and Technology, Osaka University in 2004. He joined NEC Corporation in 1986 and has been engaged in research on traffic management for high-quality and high-speed internet. His current interests include TCP session layer traffic control, network traffic traceability and network security. He is a member of IEICE. He was a secretary and has been a member of steering committee of Communication Quality Technical Group in IEICE. He is also a member of steering committee of Information Network Technical Group in IEICE. He is a vice chair person of Next Generation Network working group in 163rd Committee on Internet Technology (ITRC).

Masayuki Murata Masayuki Murata received the M.E. and D.E. degrees in Information and Computer Sciences from Osaka University, Japan, in 1984 and 1988, respectively. In April, 1984, he joined Tokyo Research Laboratory, IBM Japan, as a Researcher. From 14 September 1987 to January 1989, he was an Assistant Professor with Computation Center, Osaka University. In February 1989, he moved to the Department of Information and Computer Sciences, Faculty of Engineering Science, Osaka University. From 1992 to 1999, he was an Associate Professor in the Graduate School of Engineering Science, Osaka University, and from April 1999, he has been a Professor of Osaka University. He moved to Advanced Networked Environment Division, Cybermedia Center, Osaka University in April 2000. He has more than two hundred papers of international and domestic journals and conferences. His research interests include computer communication networks, performance modeling and evaluation. He is a member of IEEE, ACM, The Internet Society, IEICE and IPSJ.