Hierarchically Aggregated Fair Queueing (HAFQ) for Per-flow Fair Bandwidth Allocation in High Speed Networks

Ichinoshin Maki†, Hideyuki Shimonishi‡, Tutomu Murase‡, Masayuki Murata†, Hideo Miyahara†
†Graduate School of Information Science and Technology, Osaka University
‡Networking Research Labs, NEC Corporation
E-mail: i-maki@ist.osaka-u.ac.jp

Abstract—

Because of the development of recent broadband access technologies, fair service among users are becoming more important criteria. The most promising scheme of router mechanisms for providing fair service is per–flow traffic management. However, it is difficult to be implemented in high–speed core routers because per–flow state management is prohibitive; 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 propose a new traffic management scheme called Hierarchically Aggregated Fair Queueing (HAFQ) to provide per–flow fair service. Our proposed scheme can adjust flow aggregation levels according to the queue handling capability of various routers. That means the proposed scheme is scalably used 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 in estimating the number of flows, it enhances the fairness by preferentially dropping their packets. We show that our proposed scheme can provide per–flow fair service through extensive simulation and experimental studies using a network processor. Since the currently available network processors (Intel IXP1200 in our case) is not high capacity, we also give extensive discussions on the applicability of our scheme to the high–speed core routers.

I. 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. That 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 hardly provide per–flow fair service for all users [4]. 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 [5] 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 proposed 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 proposed 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 evaluated 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 proposed scheme on Intel IXP1200 network processor [6]. 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.

II. Hierarchically Aggregated Fair Queueing (HAFQ)

A. Outline

The basic mechanism of our scheme is illustrated in Figure 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 [7]. Then, the number of active flows in each queue is estimated. 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 hav-

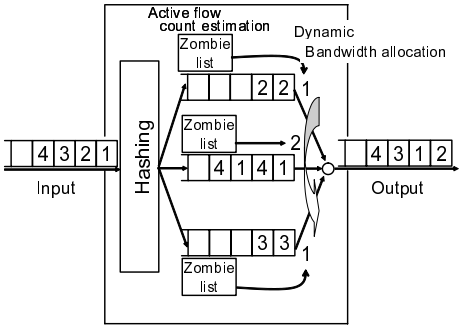


Fig. 1. Outline of the proposed scheme.

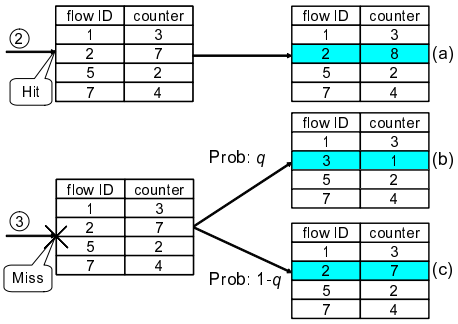


Fig. 2. Zombie list.

ing high packet arrival rates, fairness among aggregated flows in the same queue can be improved by dropping those packets preferentially.

As output operations, our proposed 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.

B. 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.
  - 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*.

Figure 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)).

C. 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.

$$\lambda_{avg} = \frac{\sum_{i=1}^{N}\lambda_i}{N} \tag{1}$$

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

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.,

$$R_i = \frac{\lambda_i}{\sum_{i=1}^{N}\lambda_i} \tag{3}$$

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 ($1 \le j \le M$) 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

$$\begin{aligned} P_1 &= (1-R_i)a + (1-R_i)(1-a)(1-R_i)a \\ &\quad + \{(1-R_i)(1-a)\}^2(1-R_i)a \\ &\quad + \cdots + \{(1-R_i)(1-a)\}^n(1-R_i)a \\ &= \frac{(1-\sum_{j=1}^{M}R_{X_j})\frac{q}{M}}{(1-\sum_{j=1}^{M}R_{X_j})\frac{q}{M}+R_i} \end{aligned} \tag{4}$$

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,

$$a = \frac{1-\sum_{j=1}^{M}R_{X_j}}{1-R_i} \times \frac{q}{M} \tag{5}$$

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

$$\begin{aligned} P_n &= R_i^{n-1}P_1 + (1-R_i)(1-a)R_i^{n-1}P_1 \\ &\quad + \{(1-R_i)(1-a)\}^2R_i^{n-1}P_1 \\ &\quad + \cdots + \{(1-R_i)(1-a)\}^nR_i^{n-1}P_1 \\ &= \frac{R_i^{n-1}(1-\sum_{j=1}^{M}R_{X_j})\frac{q}{M}}{\{(1-\sum_{j=1}^{M}R_{X_j})\frac{q}{M}+R_i\}^n} \end{aligned} \tag{6}$$

Therefore, the expectation E_i for the maximum value of the

packet counter is given by

$$E_i = \sum_{i=1}^{\infty} i\, P_i = \frac{R_i}{(1-\sum_{j=1}^{M} R_{X_j})\frac{q}{M}} + 1 \qquad (7)$$

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. (7) as follows:

$$\widetilde{R}_i = \left(1 - \sum_{j=1}^{M} R_{X_j}\right) \frac{q}{M}(\widetilde{E}_i - 1) \qquad (8)$$

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:

$$\widetilde{R}_i = (1-p)\frac{q}{M}(\widetilde{E}_i - 1) \qquad (9)$$

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

$$\begin{aligned}\widetilde{R}_{avg} &= \left\{1 - \beta\left(\frac{\widetilde{E}_i}{\widetilde{R}_i}\right)\right\}\widetilde{R}'_{avg} + \beta\left(\frac{\widetilde{E}_i}{\widetilde{R}_i}\right)\widetilde{R}_i \\ &= \left\{1 - \frac{\alpha}{1-p}\cdot\frac{\widetilde{E}_i}{\widetilde{E}_i - 1}\right\}\widetilde{R}'_{avg} + \beta\widetilde{E}_i \qquad (10)\end{aligned}$$

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. (2) and (3).

$$N = \frac{1}{\widetilde{R}_{avg}} \qquad (11)$$

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.

D. 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. (9) 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.

III. 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 [8].

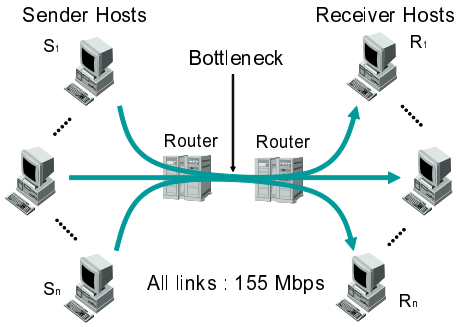


Fig. 3. Single-link model.

A. 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.

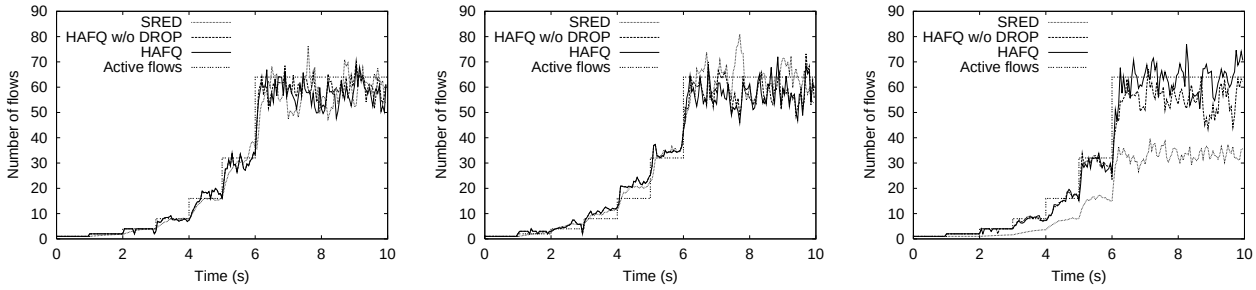
In Fig. 4(a), all flows are TCP flows and their RTTs are same. In this case, all three schemes give approximately correct numbers. In Fig. 4(b), half of the access links have 1 ms propagation delays and the other half have 0.1 ms propagation delays. This figure shows that RTTs have little influence on 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 10%. This is because packets of UDP flows are preferentially dropped and the arrival rates of all flows become more uniform.

B. 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.

Figure 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 case, the FIFO scheme gives UDP flows much more bandwidth than TCP flows and our

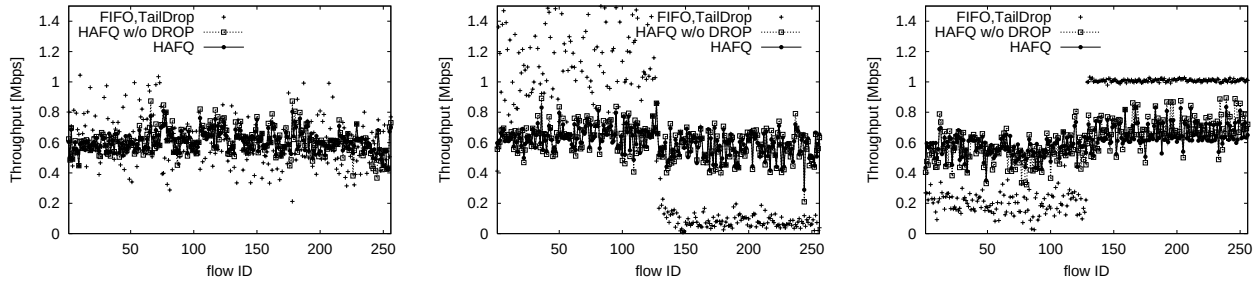


(a) TCP flows in a homogeneous environment

(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.

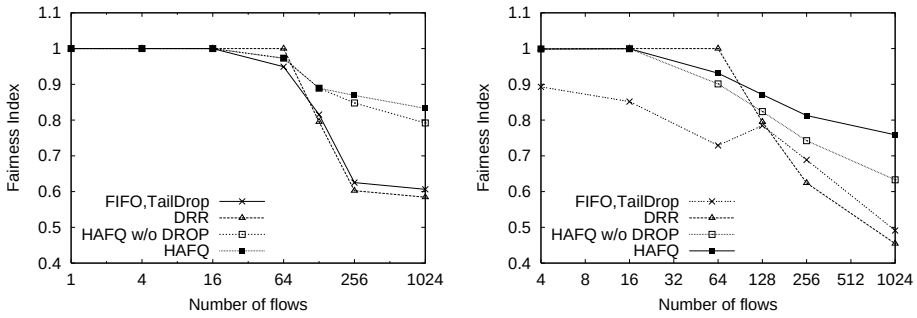


(a) TCP flows in a homogeneous environment

(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.

scheme improves fairness between TCP flows and UDP flows. In these three cases, the packet–dropping of our new scheme further improves fairness among flows.

C. 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 queues. 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 randomly for comparison purpose. 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

$$f(x_1, x_2, x_3, \cdots, x_N) = \frac{(\sum_{i=1}^{N} x_i)^2}{N \sum_{i=1}^{N} {x_i}^2} \tag{12}$$

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. 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

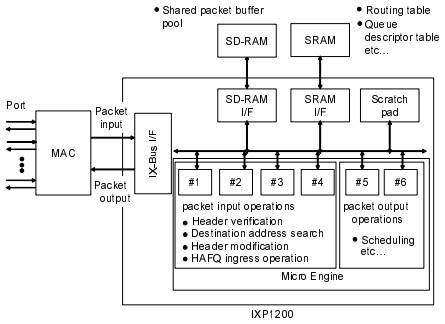


Fig. 7. Our task assignment on a network processor.

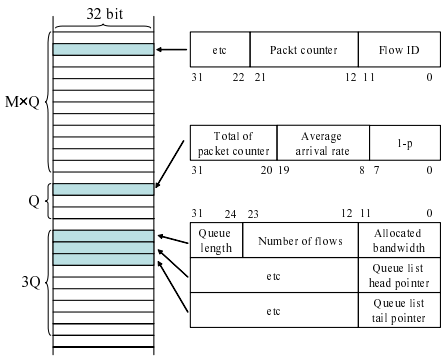


Fig. 8. Memory model for HAFQ operations.

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.

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.

IV. Implementation Design Issues of HAFQ on a Network Processor

In this section, we discuss the implementation design issues of HAFQ on the IXP1200 network processor [6], 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.

A. 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 [9]. 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 other states which are needed in performing HAFQ operations.

B. Memory Model

A memory capacity is severely limited in high–speed routers. Thus, we carefully designed the memory model of HAFQ. Figure 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. The required memory capacity for the flow count estimation and packet scheduling are $Q \times 32$ bit and $3 \times Q \times 32$ bit, respectively. Therefore, the total required memory capacity for implementing HAFQ is

$$(M + 4) \times Q \times 32 \ \ [bit] \tag{13}$$

The arguments above are not limited to specific network processors, but to a wider range of router architectures. For example, 64K queues are maintained in edge routers with 2M byte SRAM devices, and 1K queues are maintained in backbone routers with a 32K byte on–chip memory macro integrated in a queue control LSI.

C. Implementation Evaluation

We use the IXP1200 Developer Workbench [10] for evaluation. We suppose the network model shown in Figure 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.

C.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.

Figure 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.

C.2 Fairness comparison for memory requirements

We evaluate the effect of required memory capacity, i.e., implementation cost, on the fairness property. Figure 10 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

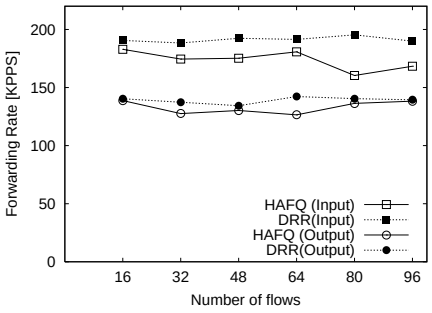


Fig. 9. Packet forwarding rate.

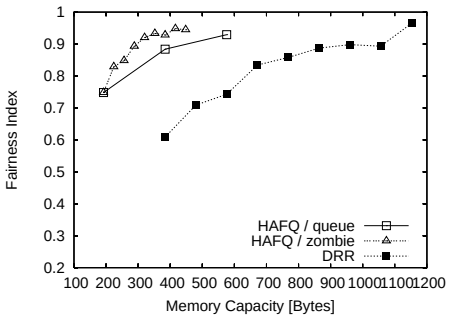


Fig. 10. Fairness in the case of the different memory capacity.

ranges from 384 byte to 1152 byte. 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, the memory requirement ranges as 192 byte to 576 byte. 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 192 byte to 448 byte.

Figure 10 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 tradeoff between memory capacity and processing performance.

Since HAFQ provides far better fairness compared to DRR when the available memory capacity is small (Fig. 10), and also the degradation of the fairness of HAFQ is smaller than that of DRR as the number of active flows increases (Fig. 6), we can conclude that HAFQ provides fair network services in high–speed routers with a small memory capacity and a large number of active flows. Note that, although a router would have several ten times more active flows compared to this evaluation, the router would be able to have several ten times more memory capacity.

C.3 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. It is too small for the router with a 10 Gbps line interface. 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 is being solved. 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 325 million and 175 million 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 against 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. (13). 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.

V. 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 proposed scheme through extensive simulation and experimental studies.

References

[1] R. Mahajan and S. Floyd, "Controlling high bandwidth flows at the congested router," tech. rep., International Computer Science Institute technical report TR-01-001, Apr. 2001.

[2] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, pp. 397–413, Aug. 1993.

[3] T. J. Ott, T. Lakshman, and L. Wong, "SRED: Stabilized RED," in *Proceedings of IEEE INFOCOM 1999*, pp. 1346–1355, Mar. 1999.

[4] M. Christiansen, K. Jeffay, D. Ott, and F. D. Smith, "Tuning RED for web traffic," *IEEE/ACM Transactions on Networking*, vol. 9, pp. 249–264, June 2001.

[5] M. Shreedhar and G. Varghese, "Efficient fair queueing using deficit round robin," *IEEE/ACM Transactions on Networking*, vol. 4, pp. 375–385, June 1996.

[6] "Intel IXP1200." available at `http://developer.intel.com/design/network/products/npfamily/ixp1200.htm`.

[7] Z. Cao, Z. Wang, and E. Zegura, "Performance of hashing–based schemes for Internet load balancing," in *Proceedings of IEEE INFOCOM 2000*, pp. 332–341, Mar. 2000.

[8] "UCB/LBNL/VINT network simulator - ns (version 2)." available at `http://www-mash.cs.berkeley.edu/ns/`.

[9] 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.

[10] "IXP1200 Developer Workbench." available at `http://www.intel.com/design/network/products/npfamily/sdk2.htm`.