Long-Term Flow Fairness with Fixed Memory Space
Jong-hwan Kim, Hyunsoo Yoon, and Ikjun Yeom, Member, IEEE

Abstract—We provide a minimal overhead FIFO queue management scheme which controls link congestion and realizes long-term flow fairness. The proposed scheme is very simple but effective to regulate long term high bandwidth flows. The proposed scheme uses light resources including additional small fixed size memory. The evaluation result verifies the max-min convergence of the proposed mechanism and shows that the proposed scheme outperforms the compared related schemes.

Index Terms—Packet-switching networks, network management, routers.

I. INTRODUCTION

In the current Internet, unfair bandwidth sharing becomes a serious problem as the amount of unresponsive traffic increases due to the growth of multimedia traffic, and as network heterogeneity increases due to the development of network technologies. Many proposals have been proposed to remove or alleviate unfairness in [1], [2]. Among the prior works for fairness, CHOKe has been paid wide attention due to its simple and effective method to penalize high bandwidth flows [1]. In CHOKe, an arriving packet is compared with a randomly selected packet from the queue, and both of them are discarded if they belong to the same flow. The main advantage of CHOKe is in its simplicity. It does not require per-flow state, and thus it is highly scalable. The main drawback of CHOKe is that the drop rate of each flow is determined by the queue occupancy of the flow. Since we cannot decouple the queue occupancy and the drop rate of a flow, to discard more packets of a high bandwidth flow, more packets of it have to be in the queue, and unfair bandwidth sharing is inevitable even though it can be alleviated by CHOKe.

In this letter, we present a very light weight queue management scheme for realizing fairness. The proposed scheme requires a fixed size of memory space. The complexity of this scheme is comparable to CHOKe, but it decouples the queue occupancy and the drop rate. To detect high bandwidth flows, this scheme relies on a cache instead of the queue. Since we can manipulate the contents of the cache whereas the queue should contain packets to be served, the proposed scheme can be more effective to realize long-term fairness than CHOKe. In a short-term interval, the proposed scheme may respond slowly due to cache update delay. However, several Internet measurement studies in [3], [4] show that the flow size and the sending rate are strongly correlated, and a small number of long-term high bandwidth flows are the major reasons of unfairness. Based on these studies, we focus on regulating long-term high bandwidth flows rather than short-term flows. Performance evaluation using ns-2 [5] shows that the proposed scheme outperforms the prior related schemes.

II. THE PROPOSED SCHEME

A. Drop policy

The proposed scheme manages a single FIFO queue to contain arriving packets at the link. In addition, it maintains a small cache to contain flow identifiers (IDs) of selective packets. When a packet arrives at the queue, the flow ID of the packet is compared with a randomly selected one from the cache with a probability \( p_d \). If the chosen flow ID from the cache is the same as that of an arriving packet, then the arriving packet is discarded. Otherwise, the arriving packet is admitted into the queue. Let \( p_m \) be the probability that a compared packet is matched with the selected one from the cache. Then, when a packet arrives at a queue, the packet is discarded with a probability, \( p \), as follows

\[
p = p_d \cdot p_m
\]

(1)

Since \( p_m \) is determined by the cache occupancies of individual flows, we have to adjust \( p_d \) to realize a desired drop rate at which the queue is not empty nor full. With a given \( p_m \), if \( p_d \) is too high, the link can suffer from low utilization due to a high drop rate. If \( p_d \) is too low, then the queue is full, and the proposed scheme could not achieve fairness since packets are forced to be dropped without differentiation.

Since \( p_d \) can be independently controlled, any prior active queue management (AQM) scheme can be employed to determine \( p_d \). In CHOKe, the RED policy [6] is used to control the average drop rate. However, recent studies on AQM [7], [8] have shown that detecting congestion only by queue length (as in RED) may not be accurate, and claimed that the packet arrival rate should be considered as well. To determine \( p_d \), in this letter, we derive a simple queue dynamics equation from the fluid model as in [7], [8].

\[
\dot{q} = r(1 - p) - C
\]

(2)

where \( \dot{q} \), \( r \), \( p \) and \( C \) are the queue length variation, the packet arrival rate, the drop rate and the link capacity, respectively. To converge the queue length to a target queue length \( q_t \) exponentially with the ratio of \( \frac{q_t}{q} \), \( \dot{q} \) should follow

\[
\dot{q} = (q_t - q)d
\]

(3)

From (2) and (3), we can find the desired drop rate \( p \) to converge the queue to the target as follows

\[
p = \frac{r - C - (q_t - q)d}{r}
\]

(4)

From (1) and (4), we can calculate \( p_d \) by \( \frac{p}{p_m} \) for \( p \leq p_m \). Note that \( p_m \) can be simply measured at a queue. If \( p > p_m \), which can happen upon severe congestion, the drop rate cannot reach
the desired $p$ even if we perform comparison with every arrival packets ($p_d = 1$). In this case, we repeat comparison several times with $p_d = 1$ to reach $p$. If $n$ number of comparisons is performed for an arriving packet, the packet drop rate becomes $1 - (1 - pm)^n \approx pm \cdot n$ for small $pm$.

### B. Cache management

Since the drop probability of a flow is determined by the cache occupancy of the flow, it is important to manage the cache to contain proper flow IDs. If we update the cache with a new arriving packet without test, the cache contains flow IDs proportional to their sending rates. Then, the proposed scheme works similar to CHOKe, and it is hard to provide an enough drop probability to a high bandwidth flow. In the proposed scheme, when a new packet is admitted at a queue, we perform one more comparison with a probability $(p_u)$ to update the cache. If the new packet and the selected one from the cache belong to the same flow, we replace the oldest flow ID in the cache with this flow ID. With this comparison based cache update scheme, the cache is updated with high bandwidth flows’ IDs with higher probabilities, and hence the cache contains high bandwidth flows’ IDs more than their sending rate proportions.

Here, the desired cache size is a just enough size to present the current traffic proportion of flows. If it is too small, it is hard to control the drop probability for fairness precisely. If it is too large, it takes long time to refresh the cache to reflect the current proportion. Considering RTTs of TCP flows in the current Internet, which are mostly less than several hundreds milliseconds, several seconds is an enough time interval to monitor steady-state sending rates of flows. Since the probability of cache update with an arriving packet is $p_m p_u$, $N p_m p_u$ number of cache updates occur in a second, where $N$ is the number of arriving packets in a second. In order for the cache to contain packets arrived in $t_s$ seconds, the cache size $C_s$ becomes $t_s N p_m p_u$. If we manage $p_u = \frac{1}{t_s p_m}$, $C_s$ becomes $N$, which maximum is $C$, where $C$ and $k$ are the link capacity and the packet size, respectively. Here note that, if $\frac{1}{t_s p_m} > 1$, we perform number of comparisons with an arriving packet. A flow ID saved in the cache memory can be defined by any combination of tuples (such as address, port etc) at a packet header. If we use 40 bytes to define a flow ID, 500 Kbyte cache size is enough to contain arrival packets’ flow IDs for $t_s$ seconds at 100 Mbps link with 1 Kbyte packet.

In order to operate properly, the proposed scheme needs to provide an opportunity for a packet to update the cache with its flow ID without comparison. For that, upon receiving a packet, we update the cache without comparison with a probability $p_s$. Here note that $p_s$ should be much less than the update probability of a flow with comparison, which is $\frac{1}{t_s}$ as discussed above, not to disturb the normal operations for fairness. We recommend to configure $p_s$ as $\frac{1}{100 t_s}$. With this configuration, 1% of the cache is updated without comparison. We look at the impact of $p_s$ in the next section.

### C. Analysis

We now analyze the proposed algorithm for the asymptotic throughput of flows with a fluid model in a single bottleneck. Suppose that there are $N$ number of flows. Let $\lambda_i$ denote the sending rate of flow $i$ where $i = 1, 2, \ldots, N$. Let $c_i$ denote the proportion of flow $i$ in the cache, and $0 < c_i < 1$. Then, the obtained throughput $\tau_i$ of flow $i$ can be calculated by

$$\tau_i = \lambda_i (1 - p_d c_i) \quad (5)$$

Here, $\lambda_i p_d c_i$ is the drop rate of flow $i$. Similarly, $p_u c_i$ is the cache update rate with flow $i$. For each cache update, the update probability with the ID of flow $i$ becomes $\frac{t_s}{t_s + \Delta c_i}$, and the probability that the ID of flow $i$ is the oldest one to be replaced is $c_i$. Therefore, the variation of $c_i$, $\Delta c_i$, upon each cache update is as follows

$$\Delta c_i = \frac{\tau_i c_i}{\sum_{j=1}^{N} \tau_j c_j} - c_i \quad (6)$$

Then, the following theorem formally states that the proposed scheme can achieve long-term fairness.

**Theorem:** In the proposed scheme, each flow competing a link can realize an equal throughput in an equilibrium state.

**Proof:** The theorem can be proved with the following three lemmas.

**Lemma 1:** If stationary equilibrium values $c_{i,0}$ and $c_{j,0}$ exist for flow $i$ and $j$, respectively, both flows have the same equilibrium throughput, i.e., $\tau_{i,0} = \tau_{j,0}$. $^2$

**Proof:** At the stationary equilibrium state, $\Delta c_i$ becomes 0, and we have $\tau_i = \sum_{j=1}^{N} \tau_j c_j \forall i \in \{1, \ldots, N\}$ from (6). This proves Lemma 1.

**Lemma 2:** If $\tau_i > \sum_{j=1}^{N} \tau_j c_j$, $\tau_i$ decreases, and if $\tau_i < \sum_{j=1}^{N} \tau_j c_j$, $\tau_i$ increases.

**Proof:** If $\tau_i > \sum_{j} \tau_j c_j$, then $\Delta c_i > 0$ from (6). Then, $c_i$ keeps increasing, and the drop rate $(p_d c_i)$ of flow $i$ also increases. Therefore, $\tau_i$ decreases. Similarly, if $\tau_i < \sum_{j} \tau_j c_j$, $\tau_i$ increases.

**Lemma 3:** $\sum_{j=1}^{N} \tau_j c_j$ converges on an equilibrium value.

**Proof:** Let $\tau_m$ and $\tau_M$ denote the minimum and the maximum throughput of all the flows, respectively. Since $\sum_{j=1}^{N} \tau_j c_j \geq \sum_{j=1}^{N} \tau_j c_j$, $\sum_{j=1}^{N} \tau_j c_j \leq \sum_{j} \tau_j c_j$, and $\sum_{j=1}^{N} \tau_j c_j = 1$, we have $\tau_m \leq \sum_{j=1}^{N} \tau_j c_j \leq \sum_{j=1}^{N} \tau_j c_j = \tau_M$. From Lemma 2, for each cache update, $\tau_m$ increases, and $\tau_M$ decreases. Therefore, $\tau_M - \tau_m$ converges on 0, and therefore, $\sum_{j=1}^{N} \tau_j c_j$ converges on an equilibrium value.

From Lemmas 1–3, we can prove that $\tau_i, \forall i \in \{1, \ldots, N\}$, converges to the same equilibrium value.

### III. Performance Evaluation

We evaluate the proposed scheme through ns-2 simulation [5]. The simulation scenario is as follows: There is a single bottleneck link with 100 Mbps. The offered traffic consists of five high-bandwidth UDP flows, 200 long-term TCP flows, and short-term TCP flows. The sending rate of each UDP flow is 4 Mbps. To simulate heterogeneity of TCP flows, RTT of the $i^{\text{th}}$ long-term TCP flow is set to $20 + 20 \times \epsilon \% 10$ msec, so that the minimum is 20 msec., and the maximum is 200 msec. Short-term TCP flows are for simulating web traffic, and they are generated as follows: For each second, we add 250 TCP flows, and they send a random number of packets. The average number of packets sent by

$^1$Note that this assumption is only for analysis, and not required for the operation of the scheme. If $c_i = 0$, flow $i$ observes no packet loss. If $c_i = 1$, the drop rate of flow $i$ becomes $p_d$.

$^2$We use $z_{i,0}$ to mean the equilibrium value of $z_i$. 

---

*Authorized licensed use limited to: Sungkyunkwan University. Downloaded on December 18, 2009 at 00:16 from IEEE Xplore. Restrictions apply.*
each flow is 20 packets. The packet size of all the flows is 1 KB.

With this scenario, we perform three sets of simulation. In the first set, we compare the results with other well-known schemes, CHOKe and RED-PD [2]. RED-PD is a preferential dropping scheme which uses packet drop history for updating high-bandwidth flow state. For queue management of the proposed scheme, we set $q_t = 0.5 \times$ queue limit, and $d = 10$. The queue limit is 400 KB. The cache size ($C_s$) is set to contain 12,500 Flow IDs, which is corresponding to $C_s = \frac{C}{T}$. If we use 40 bytes for a flow ID, then the cache size becomes 500 KB. $t_s$ and $p_s$ are set to 2 seconds and 0.005 ($= \frac{5}{1000}$), respectively. For CHOKe and RED-PD, we set the queue limit as 900 KB to include the memory used as the cache in the proposed scheme. CHOKe is set to divide queue length into 8 subregions with $2^i$ number of drop candidates for $i$-th subregion as recommended in its own literature [1]. We also follow the recommendation for RED-PD settings.

In Table I, we summarize the results. All the schemes realize high utilization, but it is observed that CHOKe fails to control high bandwidth UDP flows, and they achieve 10 times higher throughput than TCP flows. In RED-PD, even though it shows better performance than CHOKe, UDP flows still achieve much higher throughput than TCP flows. In the proposed scheme, throughput of UDP flows is effectively contained, and UDP and TCP flows achieve similar throughput. For short-term TCP flows, we present the loss rate instead of throughput since they do not send an enough number of packets to measure their throughput. It is clear that the proposed scheme protects short-term flows compared to CHOKe and RED-PD. Fig. 1 depicts throughput of the first 100 long-term TCP flows for close observation. The proposed scheme approximates the fair bandwidth allocation whereas CHOKe and RED-PD still show the tendency that the bandwidth of TCP flow is inversely proportional to RTT.

To examine the impact of $t_s$, which is the expected interval for cache refreshment, we vary it from 1 to 6 seconds under the same simulation scenario in the above. Recall that $t_s$ determines how frequently the cache is updated. Results are presented in Fig. 2. It is observed that, the proposed scheme achieves fair bandwidth among UDP and TCP flows with less than 10% deviation regardless of $t_s$, and we can confirm that the proposed scheme is not sensitive to $t_s$.

Finally, we observe the impact of $p_s$, which is the probability that a packet is on the cache without comparison. We fix $t_s$ as two seconds, and vary $p_s$ from 0.001 to 0.025. We perform the same simulation, and add a new 4 Mbps UDP flow at 80 second. In Fig. 3, we present throughput changes of the flow. At 80 second, the flow sends 500 packets per second (which corresponds to 4 Mbps), but the throughput is quickly reduced to the fair bandwidth. If $p_s$ is higher, more quickly the throughput reaches the fair bandwidth, but the differences are not significant.

IV. CONCLUSIONS

We have proposed a new AQM scheme to approximate maxmin fairness with a small additional memory. The proposed algorithm is very simple, easily implementable and has low overhead. The performance evaluation shows that the proposed scheme limits the bandwidth of unresponsive flow to the fair share of bandwidth and enforces the fairness between responsive flows with different RTTs.

REFERENCES


