WEIGHTED FAIR BANDWIDTH ALLOCATION AND ACTIVE QUEUE MANAGEMENT FOR ADAPTIVE FLOWS

Changhee Joo and Saewoong Bahk

School of Electrical Engineering and Computer Science
Seoul National University, Seoul, Korea
Email: {cjoo, sbahk}@netlab.snu.ac.kr

ABSTRACT

Some gateway algorithms have been proposed to support both fair bandwidth allocation and queue management, but they are not dealing with flow weight. We propose frame-based flow random early drop ($F^2$RED) algorithm that supports the functions of weighted fair bandwidth allocation and active queue management (AQM). Since the packet discarding behavior of AQM can disturb weighted fair bandwidth allocation, we harmonize them in $F^2$RED. Simulation results show that $F^2$RED is fair as much as Fair Queueing and also successful in queue management.

1. INTRODUCTION

End-host congestion control algorithms like TCP contribute to robustness of the Internet. The nature of distributed control makes the deployment easy and allows the network to scale up. However, the end-to-end congestion control reveals limitations as the network needs to satisfy Quality of Service (QoS) of users. Router mechanisms are introduced to improve network performance, and they are essential to meet QoS demands [1].

Gateway algorithms such as Fair Queueing [2] and its variants [3] are powerful in bandwidth allocation. The gateway with such an algorithm has a separate queue for each flow and schedules packets from all queues. This method is very effective in ensuring the fair share of link bandwidth for each flow and in bounding queueing latency of packets. However, this per-flow queueing requirement hinders the system from being deployed widely due to its complexity [4].

Another problem of per-flow queueing is that it is hard to meet requirements of the differentiated service (DiffServ) architecture, which attracts attention in providing QoS in the Internet. Two kinds of forwarding service are standardized for the architecture; Expedited Forwarding (EF) and Assured Forwarding (AF). EF is similar to a virtual leased line and AF [5] provides different levels of forwarding assurance. Since the EF service can be clearly realized by priority queue, it does not have any QoS problems. So we focus on the realization of the AF service in this paper.

The AF service requires an AQM algorithm such as Random Early Detection (RED) [6] for congestion control [5]. Flow Random Early Drop (FRED) [7] was proposed to achieve both fairness and queue management. It stochastically drops an incoming packet if a flow that the packet belongs to occupies the queue over a given threshold. It shows better fairness performance than RED without losing the advantages of queue management. However, FRED can not support the weight of a flow nor is fair as much as other schemes using per-flow queueing [4].

In this paper, we propose a gateway algorithm that supports weighted fairness in addition to queue management. To achieve these together, we modify a state-of-the-art fair queueing algorithm named Frame-based Fair Queueing (FFQ). FFQ [3] is a weighted fair queueing scheme and has the complexity of $O(1)$. It provides the worst-case latency of Fair Queueing and comparable fairness. Among Fair Queueing and its variants, we pay attention to FFQ because of its frame structure. The frame structure plays an essential role in providing fairness with low complexity. The FFQ architecture can be used in harmonizing weighted fair bandwidth allocation with packet discarding behavior of queue management.

The paper is organized as follows. We explain our proposal in section 2, which coordinates modified FFQ for weighted fairness and AQM for congestion control. In section 3, we evaluate the performance in a comparative manner. We conclude the paper in section 4.

2. FRAME-BASED FLOW RANDOM EARLY DROP ($F^2$RED)

$F^2$RED has two functions, one for weighted fairness and the other for queue management. However, both functions conflicts in their own goals. The function for weighted fair-
ness assumes that all active flows are always backlogged, but the assumption no longer holds if queue management drops packets regardless of weights. As well, queue management will not function properly if bandwidth allocation changes characteristics of aggregated flow, on which fairness and stability of queue management highly depends. In this section, we first focus on the part of weighted fairness and then, describe their coordination.

2.1. Modified FFQ

The original packet-by-packet FFQ defines the potential of a flow by the amount of normalized service received and missed by the flow during the current busy period, and the system potential as the progress of the total work done. Using the per-flow queuing, it records starting and finishing potentials of each packet of a flow. The starting potential of a packet is set by selecting the minimum of the system potential and the largest finishing potential of a flow that the packet belongs to. The finishing potential of the packet is obtained by summing its starting potential and its length divided by the flow’s service rate. The scheduler transmits a packet first that has the smallest finishing potential.

FFQ uses the frame structure to update the system potential. Since the system potential increases in proportion to time, it lags behind the total work if the sum of backlogged flow rates is smaller than the link capacity. FFQ performs periodic recalibration of the system potential to compensate for lagging. A frame is delineated by two consecutive recalibrations and we call the maximum period between recalibrations as the maximum frame size $F$. Since FFQ records the starting potential of each packet, it can recognize the end of a frame by transmitting all the packets whose starting potentials are within the maximum frame size. Upon detecting the end of a frame, FFQ calibrates the system potential to a multiple of $F$. Therefore, FFQ provides fair bandwidth allocation within the error bounded by $F$.

Since FFQ uses a drop tail queue, it lacks control for congestion, which is needed for adaptive flows like TCP. So an AQM algorithm should be accompanied by FFQ. However, it can disturb FFQ because dropping packets excessively from a flow prevents the flow from being backlogged. Since AQM calculates the packet drop probability according to the total queue or queue of flow, it can drop packet from a flow whose potential is even smaller than the system potential. AQM needs additional information about backlogged flows to harmonize with the function of weighted fair bandwidth allocation in FFQ.

In order to harmonize with AQM, F$^2$RED uses per-flow accounting instead of per-flow queuing of the original packet-by-packet FFQ. The concepts of potentials periodic recalibration are the same, but F$^2$RED uses a virtual queue for each flow to emulate the behavior of FFQ in bit-by-bit. The bit-by-bit FFQ is based on the fluid model, and transmits bits of each flow. Hence, the frame update will be invoked at the moment that bits of current frame finish service. It serves all backlogged flows in a fair manner without being affected by the packet length. Therefore, it is known to have better fairness performance than the packet-by-packet FFQ. However, it is not feasible to implement due to complexity. F$^2$RED emulates it by using per-flow accounting.

Figure 1 describes the detailed algorithm of F$^2$RED invoked by the $k$-th packet arrival of flow $i$. We assume that the service rate of the server is 1, i.e., the frame time $T$ is equal to $F$. The system potential increases linearly with time within a frame (line 3), and is recalibrated at the beginning of new frame. Comparing transmitted bits $TX$ with the frame size $FS[f]$, where $f$ is the current frame number (line 2), the algorithm checks whether the frame boundary is passed and performs the frame update (line 5-13). After the update, the starting and finishing potential of flow $i$ are calculated by using $length_b(i, k)$ and $\phi_c$ (line 16-17).

The packet is dropped stochastically according to the finishing potential of the flow and the queue length (line 18). Once the packet is accepted, the algorithm confirms $FP(i)$. Then, it looks for frame $j$ that the packet belongs to, and increases $FS[j]$ by the packet length (line 21-23). Since the starting and finishing potentials can belong to different frames, i.e., the packet lies on a frame boundary, the counter $FS[j]$ increases by $length_f(i, k)$ which is a fraction of the packet length in bits belonging to frame $j$.

Since the algorithm does not refer to other flows’ states in updating the flow $i$’s potential and the system potential, it has the complexity of $O(1)$ like the original packet-by-packet FFQ.

2.2. Coordinating FFQ and AQM

To decouple congestion control for queue management from flow control for fairness, F$^2$RED uses an AQM algorithm that works on the aggregated flow level in managing queue length [1]. The acceptance of a packet is decided by FFQ and AQM independently. While FFQ calculates drop probability $prob_{ffq}$ from the fair share of a flow that the packet belongs to, AQM decides drop probability $prob_{red}$ from queue length and the relative weight of the flow. The frame structure of FFQ assists AQM in calculating relative weights and the gateway discards the packet with the smaller probability of these two.

The detailed algorithm is given at the end in Fig. 1. FFQ calculates the $prob_{ffq}$ by a RED-like mapping function with two potential thresholds $min_p$ and $max_p$ at line 4. Finishing potential $FP(i)$ mainly affects the $prob_{ffq}$. $P_{slope}$ determines the slope of the linear mapping function and it is equivalent to the maximum probability $P_{max}$ of RED. The $prob_{red}$ comes from the RED’s mapping function $f_{red}(qlen)$. Since the original RED drops a packet without considering
the weight of a flow, it overestimates the drop probability of a flow whose relative weight is larger than \( \frac{1}{N[f]} \), where \( N[f] \) is the number of active flows in frame \( f \). Hence, the probability needs to reflect relative flow weights to achieve weighted fairness.

Without compensation, flow \( i \) will have \( \frac{1}{N[f]} \) rather than its fair share \( \frac{\phi_i}{\text{weights}[f]} \), where \( \text{weight}[f] = \sum_m \phi_m \) and \( m \in F(f) \) and \( F(f) \) is the set of active flows in frame \( f \). Hence, the rate of flow \( i \) should be compensated by \( N[f] \frac{\phi_i}{\text{weights}[f]} \). Since the TCP rate is inversely proportional to the square of drop probability [8], the \( \text{prob}_{\text{red}} \) can be compensated by the factor \( \left( \frac{\text{weights}[f]}{N[f]} \right)^2 \). Then, the packet is discarded with the smaller probability of either \( \text{prob}_{\text{ffq}} \) or \( \text{prob}_{\text{red}} \).

Therefore, FFQ does not influence AQM’s decision unless the flow receives more than its fair share.

\[
\text{weights}[f] \quad \text{and} \quad N[f]
\]

can be updated on every packet acceptance, still requiring the complexity of \( O(1) \) to process a packet. We put the following after line 23,

\[
\text{weights}[f] \quad \text{and} \quad N[f]
\]

where the function \( \text{int}(x) \) returns the integer of \( x \).

3. SIMULATIONS

We use the network simulator ns-2 [9] to evaluate our proposal by comparing it with RED and FRED\(^1\). The simulations are based on a dumbbell topology in which all connections traverse a single bottleneck link with the capacity of 10 Mbps. The bottleneck link queue operates in byte mode with the buffer size of 300 KB. All connections have identical round-trip time (RTT) of 40 ms and the packet size of 500 bytes. The parameters of RED are configured as follows: \( \text{min}_{th} = 62.5 \) KB, \( \text{max}_{th} = 187.5 \) KB, and \( P_{\text{max}} = 0.005 \). \( w_0 \) is automatically calculated from RTT and the link bandwidth. The configurations of RED and FRED are similar to that of RED. The potential thresholds \( \text{min}_{th} \) and \( \text{max}_{th} \) are set so that the total frame bits below each threshold are equal to \( \text{min}_{th} \) and \( \text{max}_{th} \) respectively. The potential slope \( P_{\text{slope}} \) is set to \( P_{\text{max}} \).

3.1. Fairness

We first examine the weighted fairness of CBR traffic. Establishing 16 UDP connections, we generated CBR traf-
Figure 2 illustrates bandwidth allocation among 16 connections. It shows the fairness performances of the competitive schemes. Since RED does not consider the connection weights, it allocated bandwidth in proportion to the transmitting rates. On the contrary, FRED achieved even distribution of bandwidth as expected. FRED also allocated bandwidth fairly but not as much as FRED. Connection 1 of the smallest rate got less than its fair share.

In order to clearly demonstrate the difference, we compare the fairness quotient by changing the number of connections $N$ in simulations. The fairness quotient $Q_F$ is defined in [10] as,

$$Q_F = \left( \frac{\sum_{i=1}^{N} b_i}{\sum_{i=1}^{N} b_i^2} \right)^2,$$

where $b_i$ is the bandwidth obtained by flow $i$. The quotient gets closer to 1 as the bandwidth is allocated more fairly, and becomes $\frac{1}{N}$ if all the bandwidth is dedicated to a single flow. Figure 3 shows the fairness quotient with varying $N$ and ensures that FRED slightly outperforms RED and RED.

In the second experiment, we establish 16 TCP connections with linearly increasing weights. Ideally, the allocated bandwidth should be linearly increased with the connection number assuming all the TCP connections are greedy FTPs. Since RED and FRED cannot give a weight to each flow, they were unable to allocate linearly increasing bandwidth as shown in Fig. 4.

Since the quotient $Q_F$ can not represent the degree of weighted fairness appropriately, we define the weighted fairness quotient $Q_W$ as the measure by modifying $Q_F$.

$$Q_W = \left( \frac{\sum_{i=1}^{N} b_i/\phi_i}{\sum_{i=1}^{N} (b_i/\phi_i)^2} \right)^2.$$

The weighted quotient $Q_W$ also has a value in $[1, \frac{1}{N}]$.

The simulation results with various numbers of TCP connections $N$ are shown in Fig. 5. The quotients of RED and FRED decrease with the increase of $N$ because they evenly allocate bandwidth while those of FRED are close to 1 ensuring their support of weighted fairness.

For performance evaluation of fairness according to RTT, we set the weights for all the flows identical, and establish 16 TCP connections whose RTT are linearly increasing according to the connection number, i.e., given by $i \times 16$ms for flow $i$. It is well known that a TCP connection with shorter RTT takes more bandwidth. Figure 6 presents the bandwidth allocation among the connections. RED shows the
worst performance. FRED outperforms RED but still shows the same tendency. $F^2$RED removes the bias by allocating the bandwidth almost evenly. The fairness quotients of these schemes clearly demonstrate the performance: 0.9222 for RED, 0.9701 for FRED, and 0.9973 for $F^2$RED.

### 3.2. Queue management

The gateway using AQM is expected to control congestion to provide the AF service. From the viewpoint of RED, the queue length should lie in between $min_{th}$ and $max_{th}$ in order to prevent low utilization or buffer overflows when traffic load changes. Figure 7 illustrates the queue length variations of AQM algorithms. Eight TCP connections are established at the beginning and additional eight connections are added every 30 second up to 32 connections. The weights of connections are linearly increasing for $F^2$RED.

Since RED uses its queue length as the congestion measure, it got larger queue length with traffic increase as shown in Fig. 7 (a). It showed oscillatory behavior when 32 TCP connections were established after 100 second. FRED and $F^2$RED showed better performance in queue management. Unlike RED, they succeeded in keeping the queue length between $min_{th}$ and $max_{th}$ without oscillation. This confirms that a fairness scheme can enhance the congestion control capability of queue management. Note that $F^2$RED allocated linearly increasing bandwidth according to the weight of each flow while FRED did not consider the weight.

### 4. CONCLUSION

We proposed a gateway mechanism named $F^2$RED that provides weighted fair bandwidth allocation while succeeding in queue management. $F^2$RED achieved weighted fairness by modifying FFQ, and made use of its frame structure to harmonize the packet discarding behaviors of the fairness scheme and the AQM scheme.

Simulation results confirmed that $F^2$RED is successful in providing weighted fairness and managing the queue length at the same time. On the contrary, RED and FRED could not support the weight for each flow. When all weights are identical and connections have different RTTs, $F^2$RED also outperformed RED and FRED in terms of fairness.

### 5. REFERENCES


