A Novel Scheme for Improving the Fairness of Queue Management in Internet Congestion Control

Jungmin Lee, and Kwangsue Chung

Communications Engineering Department, Kwangwoon University, Seoul, Korea

Summary
This paper proposes a novel scheme to improve the fairness of active queue management congestion control. In order to reduce the increasing rates of packet loss caused by the exponential increase of network traffic, IETF (Internet Engineering Task Force) is considering the deployment of active queue management techniques such as RED (Random Early Detection). RED is a simple scheme, but it does not support the fairness of traffic protection from misbehaving flows, such as short RTT (Round-Trip Time) TCP flows and non-TCP flows, i.e., UDP flows.
To solve this problem, we propose a simple fairness queue management scheme, called the RED with AFQM (Approximate Fair Queue Management) scheme, which discriminates against flows that submit more packets than their shares allocated via a router. By doing this, the scheme aims to ensure the fairest queueing policy. In addition, the RED with AFQM scheme can be easily implemented with small overhead.

Key words:
AFQM, active queue management, congestion control, fairness, RED

1. Introduction

Today, the Internet protocol architecture is based on connectionless end-to-end packet service using IP protocol that has demonstrated advantages such as connectionless design, flexibility and robustness; however, this design is difficult to control. Therefore, a carefully designed solution is required to provide efficient and effective services under heavy loads. In fact, a lack of attention to the dynamics of packet forwarding can result in service degradation. This phenomenon was first observed during the early growth phase of the Internet in the mid-1980s and is technically called "congestion collapse" [1].

Especially, the current system depends on congestion avoidance mechanisms implemented in transport layer protocols, like TCP, to provide good services under heavy loads. However, a lot of TCP implementations do not include congestion avoidance mechanisms, either deliberately or by accident [2], [3].

It has become clear that the TCP congestion avoidance mechanisms, while necessary and powerful, are not sufficient to provide good services in all circumstances. Basically, there is a limit to how much control can be achieved at the edges of the network. Some router mechanisms are needed to complement the endpoint congestion avoidance mechanisms [4].

In order to perform congestion control based on IETF Standards, it is useful to distinguish between two classes of router algorithms related to congestion control: "queue management algorithms" versus "scheduling algorithms". All router algorithms (scheduling or queue management) developed thus far have been able to either provide fairness or simple implementation, but they cannot provide both simultaneously. This has led to the belief that the latter two goals are somewhat incompatible [1].

This paper takes a step in the direction of bridging the gap between fairness and simplicity. Specifically, we propose a novel scheme that improves the fairness of the existing active queue management algorithm, called RED with AFQM (Approximate Fair Queue Management), that discriminates against flows that submit more packets than their shares allocated by a router. By doing this, the scheme aims at making the approximate policy of fair queueing. The rest of the paper is organized as follows. Section 2 introduces existing congestion control approaches and describes RED. Section 3 presents the AFQM scheme. The simulation results are presented in Section 4. Section 5 concludes this paper.

2. Related Work

In order to solve the fairness and congestion problems of networks, it is necessary to have router mechanisms that shield responsive flows (TCP friendly flows) from non-TCP flows or misbehaving flows, and it is necessary to provide good quality of service (QoS) to all users. As discussed in IETF, there are two types of router algorithms for achieving congestion control. The generic scheduling algorithm, exemplified by the well-known Fair Queueing (FQ), requires the buffer at each output of a router to be partitioned into separate queues each of which will buffer the packets of one of the flows. Packets from the flow buffers are placed on the outgoing line by a scheduler according to an approximate bit-by-bit, round-robin

Manuscript received August 5, 2011
Manuscript revised August 20, 2011

discipline. However, it is well known that this approach requires complicated per flow state information, making it too expensive to be widely deployed [5], [6].

To reduce the cost of maintaining flow state information, Stoica et al. have recently proposed a scheduling algorithm called the Core Stateless Fair Queueing (CSFQ) [7]. The goal of CSFQ is to achieve fair queueing without using per-flow state in the core of the router island. On entering the network and the packets are marked with an estimate of their current sending rates. A core router estimates a flow's fair share and preferentially drops a packet from the flow based on the fair share and the rate estimate carried by the packet. Thus, an edge router holds onto the per flow state information and estimates each flow's arrival rate. These estimates are inserted into the packet headers and passed on to the core routers. This scheme reduces the core router design complexity and overhead. However, the edge router design is still complicated. Moreover, because of the rate information in the header, core routers have to extract packet information differently from traditional routers. Another notable scheme that aims to approximate FQ at a smaller implementation cost is Stochastic Fair Queueing (SFQ) proposed by McKenny. SFQ classifies packets into a smaller number of queues than FQ using a hash function [8]. Although this scheme reduces FQ design complexity and overhead, SFQ still requires around 1000 to 2000 queues in a typical router to approach the FQ performance.

Thus, scheduling algorithms can provide fair bandwidth allocation, but they are often too complex for high-speed implementation and do not scale well to a large number of users. Also, the algorithmic complexity and state requirements of scheduling make the deployment of scheduling algorithms difficult. On the other hand, queue management algorithms have had a simple design from the outset. Given their simplicity, the goal is achieving approximate fairness. This approach, called Active Queue Management (AQM), uses advanced packet queuing disciplines outside of the traditional FIFO drop-tail queueing on an outbound queue of a router to actively handle (or avoid) congestion with the help of cooperative traffic sources [1], [9].

In the Internet, TCP recognizes packet loss as an indicator of network congestion, and the back-off algorithm reduces the transmission load when network congestion is detected. One of the earliest and well-known AQM mechanisms is Random Early Detection (RED), which prevents congestion via monitoring the outbound buffers to detect impending congestion, and RED randomly chooses and notifies senders of network congestion so that they can reduce their transmission rates. The drop probability increases with the level of congestion. Since RED acts in anticipation of congestion, it does not suffer from the "lock out" and "full queue" problems inherent in the widely deployed Drop Tail mechanism. By keeping the average queue-size small, RED reduces the delays experienced by most flows [10].

However, like Drop Tail, RED is unable to penalize non-TCP flows. This is because the percentage of packets dropped from each flow over a period of time is almost the same. While fairly handling congestion for TCP friendly flows, RED has a potentially critical problem that non-TCP flows that are unresponsive or have greedier flow-control mechanisms than TCP can take a greater share of the output bandwidth than TCP flows. In the worst case, it is possible for non-TCP flows, especially the unresponsive ones, to monopolize the output bandwidth while TCP connections are forced to transmit at their minimum rates. This unfairness occurs because non-TCP flows reduce transmission loads to be relatively less than TCP friendly flows or do not reduce at all, and the same drop rate is applied to every flow. Consequently, non-TCP flows or misbehaving flows can take up a large percentage of link bandwidth and starve out TCP friendly flows.

RED also provides little protection from high-bandwidth flows, such as robust TCP flows, that consume excessive bandwidth at the expense of other flows at the router. These high-bandwidth flows can be particular TCP flows with short RTT (Round Trip Times) or more problematic flows using different end-to-end congestion controls. During congestion, it is important to control the high-bandwidth flows to ensure the performance of the rest of the traffic [11- 15].

In addressing the fairness problem, there have been strong arguments that non-TCP flows or misbehaving flows should be penalized to protect well-behaved TCP flows. A few variants such as RED with a penalty box and Flow Random Early Drop (FRED) have been proposed to improve RED's ability to distinguish unresponsive users. FRED is an active queue management approach that incorporates this argument. FRED adds per-active-flow accounting to RED by isolating each flow from the effects of other flows. FRED enforces fairness in terms of output buffer space by strictly penalizing non-TCP flows or misbehaving flows to ensure that all have an equally fair share while assuring packets from flows that do not consume their fair share are transmitted without loss. FRED not only achieves its purpose of protecting TCP flows from non-TCP flows or misbehaving flows, but it also protects fragile TCP connections from robust TCP connections [13]. However, per-active-flow accounting is expensive and might not scale well.

These variants incur extra implementation overhead since they need to collect certain types of state information. RED with a penalty box stores information about unfriendly flows while FRED needs information about active connections. Ott et al. propose an interesting algorithm called Stabilized RED (SRED) that stabilizes the occupancy of the FIFO buffer independently of the number of active flows [14]. More interestingly, SRED

estimates the number of active connections and finds candidates for misbehaving flows. SRED does this by maintaining a data structure, called the "Zombie list", that serves as a proxy for information about recently seen flows.

Although SRED identifies misbehaving flows, it does not propose a simple router mechanism for penalizing misbehaving flows. Pan et al. proposed a notable scheme, CHOKe (CHOose and Keep for responsive flows, CHOose and Kill for unresponsive flows), that aims to approximate fair bandwidth allocation at a smaller implementation cost. An incoming packet is matched against a random packet in the queue. If these two matched packets belong to the same flow, then both packets are dropped. Otherwise, the incoming packet is admitted with a certain probability. The rationale behind this scheme is that non-TCP flows or misbehaving flows are likely to have more packets in the queue [15]. CHOKe is not likely to perform well when the number of flows is large (compared to the buffer space), and even non-TCP flows or misbehaving flows have only a few packets in the queue. The simulations show that CHOKe achieves limited performance; for example, in the simulations, the high-bandwidth UDP flows gets much more than their fair share. The recent paper by Yang et al. proposes an interesting algorithm called Enhanced-Fairness RED (EF-RED), which modifies the formulas of calculating the final packet-discard probability and packet count of the RED algorithm based on the sending rate formula of TCP flows [16]. EF-RED only considers the bandwidth fairness problem caused by the different packet sizes, and it does not improve other factors affecting the bandwidth allocation in congested networks.

In summary, all router algorithms (scheduling and queue management) developed thus far have been either able to provide fairness or simple implementation, but they do not provide both simultaneously. This has led to the belief that the two goals are somewhat incompatible.

This paper takes a step in the direction of bridging fairness and simplicity. Specifically, we propose an active queue management algorithm, called RED with AFQM (Approximate Fair Queue Management), which is simple to implement (since it requires little state information) and differentially penalizes misbehaving flows by dropping more of their packets. By doing this, AFQM aims to approximate max-min fairness for flows that pass through a congested router.

3. RED with AFQM

In this paper, the novel queue management scheme, called AFQM, is proposed to improve the fairness of existing AQM algorithms. This scheme discriminates against the flows that submit more packets than their allowed fare share. AFQM provides selective dropping using fairness adjustment parameters based on per-active-flow buffer counts during a certain period. Fig. 1 shows the structure of AFQM that inter-networks with existing AQM algorithms in a router. In order to improve the fairness of existing AQM, AFQM has the selective dropping scheme for non-fairness connections such as non-TCP flows or misbehaving flows at congested output links.

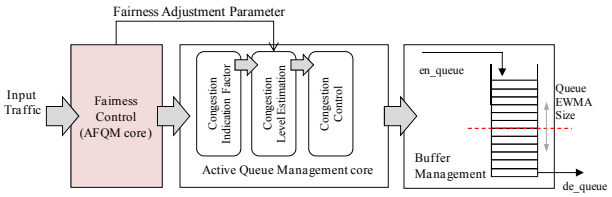

Fig. 1. The Approximate Fair Queue Management Structure

3.1 Fairness Adjustment Parameter

We define the fairness share rate ($rate_{fair}$) that is satisfied in the fair queueing policy. We have the following expressions for $rate_{fair}$ as shown in (1), as a function of the sending rate of connection i ($rate_i$) and fairness adjustment parameter (P_{fa}). If $rate_i$ is not shared as the fairness share rate, then P_{fa} controls $rate_i$ to make the approximate fair queueing policy.

$$rate_{fair} = rate_i \times \left(1 - P_{fa}\right) \tag{1}$$

The $rate_{fair}$ for congested links can also be expressed in terms of BW and n as shown in (2), where BW is the bandwidth of the output link and n is the number of active flows during a certain period (ΔT)

$$rate_{fair} = \frac{BW}{n} \tag{2}$$

The $rate_i$ can now be written as (3), where S is the packet size and N_i is the number of packets buffered in a router queue during ΔT for connection i.

$$rate_i = \frac{\left(S \times N_i\right)}{\Delta T} \tag{3}$$

The ΔT in AFQM can now be written as (4), where N_T is the total number of packets buffered in a router queue during ΔT .

$$\Delta T = \frac{(S \times N_T)}{BW} \quad (4)$$

The P_{fa} has been derived as the (5) using (2), (3), (4).

$$P_{fa} = 1 - \frac{N_T}{(N_i \times n)} \quad (5)$$

The AFQM can ensure fairness for each flow by using the fairness adjustment parameter. Therefore, the AQM algorithm with the fair adjustment parameter can control or penalize the flows that submit more packets than their fair share allowed.

Existing schemes such as FRED, CHOKe and SFB require flow state maintenance and complex per-flow processing work. However, the AFQM has a simple management mechanism that is periodic event (update time ΔT) based flow state maintenance with a reduction in the work complexity. The AFQM is easy to implement with smaller overhead than that of existing schemes.

3.2 Fairness Control

Our proposed AFQM scheme involves placing a pre-filter in front of the AQM such as RED. To manage fairness for each flow in a congested router, we propose RED with the AFQM algorithm; it is a modified version of RED or an additional function module to improve the fairness of RED. RED with the AFQM mechanism has an additional congestion level estimation mechanism instead of a simple congestion level estimation by queue size before the AQM process congestion control such as randomly chosen packet dropping. This mechanism includes a selective dropping method that uses fair adjustment parameters based on per-active-connections; it can indicate the congestion of filtered connections that have non-fairness sending rates. To indicate the unfairness of filtered connections, RED with the AFQM operates, as shown in Fig. 2, before the congestion control takes place by using AQM.

In Congestion Situation (Calculation P_{fa} for flow i)

If ($P_{fa} \le 0$)
    No Packet Drop
Else
    Active Queue Management (RED) with P_{fa}

Fig. 2. Active Queue Management (RED) with AFQM mechanism

RED with the AFQM maintains a count of buffered packets N_i and N_T for connection i that currently has some packets buffered in a router queue during ΔT. The RED with AFQM calculates the fairness adjustment parameter, as in (5), using N_i, N_T and n values. If the receiving rate of connection i ($rate_i$) is less than the fairness share rate ($rate_{fair}$) given in the fairness policy, then the fairness adjustment parameter (P_{fa}) has a negative value. If $rate_i$ is greater than $rate_{fair}$, then P_{fa} has a positive value. The RED with AFQM scheme uses P_{fa} as the parameter to determine whether or not to accept a packet into the queue in a router from a non-fairness congestion situation, as shown in Fig. 2, before the congestion control is implemented by AQM. The incoming packet is always accepted (no drop) if P_{fa} has a negative value. In other words, without packet dropping, the RED with AFQM forwards packets belonging to the connection with P_{fa} < 0. If the packets belong to the connection with P_{fa} > 0, then the exceeded packets are subject to RED's random drop probability in proportion to P_{fa}.

If unfairness connections share a bottleneck link under the ideal FQ, then a connection that consumes less than the fair share rate should have no more than one packet queued. With a single FIFO or generic AQM such as RED, this condition does not hold, and the number of backlogged packets for low bandwidth connections should still be small. However, RED with AFQM manages the connections to ensure approximate fairness as part of the fair queueing policy. Fig. 3 shows the inter-operation pseudo code of the RED with AFQM algorithm.

Active Queue Management (RED) with P_{fa}

Calculate (EWMA) avg_q
If ($avg_q \le \min_{th}$)
    No Packet Drop
Else If ($\min_{th} \le avg_q \le \max_{th}$)
    Calculate P
    RED with AFQM $P_{new} = P + P_{fa}$
Else If ($\max_{th} \le avg_q$)
    Drop the Arriving Packet

Fig. 3. The inter-operation pseudo code of the RED with AFQM

To control congestion in a router, the RED with AFQM calculates the average occupancy of the FIFO buffer using

EWMA (exponential weight moving average), which is generally similar to what RED does. The RED with AFQM also marks two thresholds on the buffer; a minimum threshold ($\min_{th}$) and a maximum threshold ($\max_{th}$). If the average queue size (avg_q) is less than $\min_{th}$, then every arriving packet is queued into the FIFO buffer. If the aggregated arrival rate is smaller than the output link capacity, then the avg_q should not build up very often and packets are not dropped frequently. If the average queue size is greater than $\max_{th}$, then every arriving packet is dropped. This moves the queue occupancy back to below $\max_{th}$. The packet drop probability (P) that depends on the average queue size is computed in the exact same way as that of RED. An incoming packet is always accepted if the connection has $P_{fa} < 0$ packets buffered or the avg_q is less than $\min_{th}$. When $P_{fa} > 0$ and the avg_q is bigger than $\min_{th}$, each arriving packet is calculated as a new packet drop probability (P_{new}) using P_{fa} to perform congestion control. On the other hand, the packet drop probability has to be increased by P_{fa} when P_{fa} is larger than zero (0).

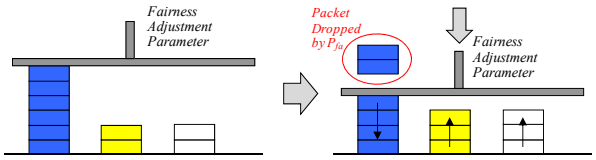

Fig. 4. Process of the fairness adjustment parameter operation

We describe an algorithm, RED with AFQM, which can differentially penalize non-TCP flows and misbehaving flows to serve well-behaving TCP. In the simplest form, shown in Fig. 4, this process, which has a per-connection fairness indication, can control the unfair connection. For connections with $P_{fa} > 0$, the connections will control congestion in a router by both the packet drop probability and the fairness adjustment parameters. Otherwise, for $P_{fa} < 0$, the connections will not control congestion. These connections, which are $P_{fa} < 0$, can better grab the bandwidth than other connections.

In the process, this increases the available bandwidth by penalizing unfair connections and decreases the packet drop rate for the connections with $P_{fa} < 0$. The connection that is $P_{fa} < 0$ can better grab the bandwidth than other connections. For example, when the high-bandwidth connection in Fig. 4 is penalizing to make the fair share rate by using P_{fa}, this decreases each packet drop rate for the rest of the connections, and it allows low-bandwidth connections ($rate_i$ is less than $rate_{fair}$)to grab the available bandwidth at the congested link if those connections have sufficient demand to grab the available bandwidth.

In summary, if incoming packets belong to a connection with $P_{fa} < 0$, then the RED with AFQM forwards the packets to the output link without any dropping. If the packets belong to a connection with $P_{fa} > 0$, then the exceeded packets are subject to congestion control. This congestion control is executed along the RED's random drop probability in proportion to P_{fa}.

3.3 ΔT of AFQM scheme

The P_{fa} of RED with AFQM uses a proportion of buffered packets for each connection during ΔT. The ΔT in the AFQM scheme is a very important parameter because the accuracy of P_{fa} depends on a period of collecting information, and the fairness of RED with AFQM is controlled by P_{fa}. In order to make accurate observations for an unfair situation at a congested router, the ΔT can be larger than the maximum queueing delay of the buffered packets. Equation (6) satisfies this condition.

$$\Delta T \geq \frac{B}{BW} \tag{6}$$

In general, the buffer space of routers in real backbones is based on the bandwidth-delay product. The router buffer space with a bandwidth-delay product can now be written as (7), where Delay (RTT_a) is a set up suitable network delay of the serving network by the administrator of the network service provider and B is the router queue size [17].

$$B = BW \times Delay(RTT_a) \tag{7}$$

The ΔT of RED with AFQM has been derived as (8) using (7).

$$\Delta T \geq Delay(RTT_a) \tag{8}$$

The bandwidth-delay product of the link is applied to protocol engines that implement flow/error control and

need to receive ACks (acknowledgements) before they can go over.

3.3 The AFQM scheme overhead

The well-known FQ algorithm not only has to manage active flows but also non-active flows during a specific time. Moreover, the FQ requires the buffer at each router output to be partitioned into separate queues each of which will buffer the packets of one of the flows. However, the RED with AFQM has simplicity that requires a single queue for the buffer at each router output while the RED with AFQM's overhead depends on the number of active flows during ΔT. To calculate the fairness adjustment parameter (P_{fa}), the RED with AFQM has to know the number of active flows and the number of buffered packets for each active flow. Therefore, the overhead of RED with AFQM is in proportion to the number of active flows during ΔT. To know the overhead of RED with AFQM, we simulate a simple test. When the single queue in a router is shared by 180 TCP flows being sent step-wise during 1000 seconds, Fig. 5 shows the number of active flows in a router during ΔT.

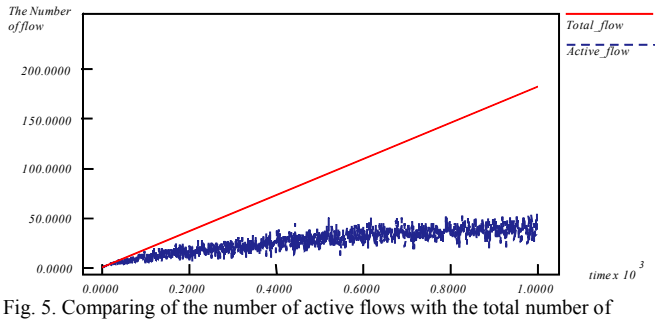

Fig. 5. Comparing of the number of active flows with the total number of flows to calculate AFQM overhead

From Fig. 5, the number of active flows in a router during ΔT is not the number of total flows that pass through a router. Accordingly, as shown in Fig. 5, the overhead of RED with AFQM is not directly in proportion to the number of active flows. First of all, the RED with AFQM requires only a signal queue for the buffer at each router output; it does not require multiple queues to achieve fairness for each flow. It is easy to implement RED with AFQM with a small overhead.

4. Simulations and Evaluation

This section presents the performance simulation results of RED with AFQM in penalizing misbehaving flows, and thus approximating fair bandwidth allocation on congested routers. In order to evaluate the performance of RED with AFQM, a number of experiments have been performed on the basis of NS (Network Simulator) of LBNL (Lawrence Berkeley National Laboratory) [18]. The results are presented as follows:

- First, compare responsive flows with unresponsive flows
- Second, compare fragile flows with robust flows
- Third, compare the transport layer protocols

4.1 Responsive Flows vs. Unresponsive Flows

In this section, we ran an NS simulation for the RED, FRED, the RED with AFQM and the FQ schemes to compare them in terms of the effects of queue management mechanisms on network bandwidth allocation fairness of the router. Every simulation had the exact same settings, except for the network routers, each of which was set to use one of the above two outbound queue management mechanisms. The network topology is shown in Fig. 6.

The congested link in this network is between routers R1 and R2. The link, with a capacity of 15Mbps (propagation delay of 20ms), is shared by m TCP (responsive) flows and n UDP (unresponsive) flows. An end host is connected to the routers using a 10Mbps link, and the end host links have a small propagation delay of 10ms.

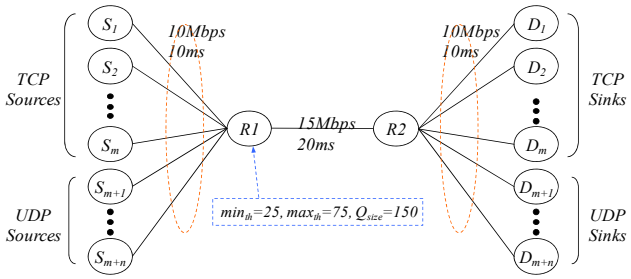

Fig. 6. Network configuration to evaluate the performance of the RED with AFQM algorithm

In the first simulation scenario, the rate for the UDP source is 1Mbps. A simulation configuration with 10 TCP and 1 ~ 10 UDP sources is designed for the test. As known, if a mixture of responsive and unresponsive flows shares a bottleneck link, then severe unfairness is likely to occur as the unresponsive flows grab the bandwidth because unresponsive flows do not reduce their transmission load for congestion control. We measured the performance of RED, FRED, RED with AFQM and FQ in terms of fairness. We compare the average per-flow throughput of each responsive, i.e., unresponsive class, which is the average aggregated class throughput divided by the number of flows in the class, to visualize how fairly the

output bandwidth is assigned to each class considering the number of flows in the class. The simulation results are summarized in Table 1.

Table 1. Aggregated Class Throughput at the Congested Link: 10 TCP flows with 1~10 UDP flows

Unit: [Mbps]

# UDP flow	TCP (FRED)	UDP (FRED)	Total (FRED)	TCP (RED)	UDP (RED)	Total (RED)	TCP (AFQM)	UDP (AFQM)	Total (AFQM)
1	12.32	1.00	13.31	12.54	0.99	13.53	12.74	0.72	13.46
2	10.66	1.99	12.65	10.93	1.96	12.89	11.48	1.41	12.89
3	9.27	2.93	12.20	9.31	2.92	12.23	10.43	2.00	12.43
4	8.12	3.79	11.92	8.03	3.85	11.88	9.42	2.59	12.01
5	7.21	4.53	11.74	6.81	4.76	11.57	8.79	3.10	11.89
6	6.24	5.32	11.56	5.69	5.67	11.36	8.04	3.65	11.69
7	5.45	5.95	11.40	4.68	6.54	11.22	7.24	4.29	11.53
8	4.84	6.49	11.33	3.76	7.41	11.17	6.55	4.82	11.37
9	4.28	6.99	11.27	2.97	8.18	11.15	5.87	5.46	11.33
10	3.84	7.41	11.25	2.30	8.85	11.15	5.20	6.04	11.24

From Table 1, we can clearly see that the RED does not discriminate against unresponsive flows. The UDP flows (10 UDP sources) take more than 85% of the bottleneck link capacity, and the TCP connection (10 TCP sources) can only take the remaining 15% of the bottleneck link capacity. The RED with AFQM, on the other hand, improves the throughput of the TCP flows by limiting the UDP throughput to 50% of the bottleneck link capacity. To gauge the degree to which the RED with AFQM achieves fair bandwidth allocation, the average aggregated class throughput in the simulation above, along with their ideal fair shares, are plotted in Fig. 7.

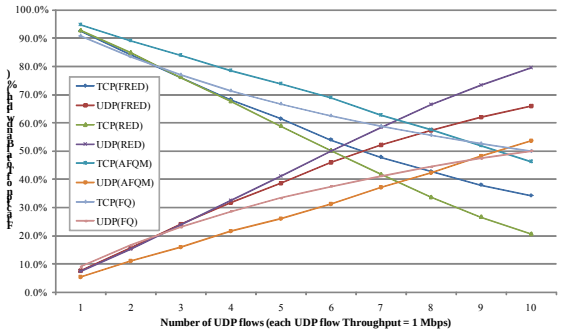

Fig. 7. Fairness comparison between the RED with AFQM, FQ, RED and FRED (10 TCP, 1~10UDP)

In the next simulation scenario, we vary the UDP arrival rate r to study the AFQM performance under different traffic load conditions. A simulation configuration with 10 TCP and 1 UDP sources is designed for the test. The UDP source sends packets at a rate r Kbps, where r is a variable. The simulation results are summarized in Table 2.

Table 2 lists the throughput of the aggregated classes in the network. From the simulation results in Table 2, the conclusion is that the RED does not discriminate against unresponsive flows; it is obvious that the RED cannot regulate greedy connections. The unresponsive flows use up all network bandwidth and starve out well-behaving flows. AFQM, on the other hand, dramatically improves the TCP flows throughput by limiting the UDP throughput to 10% of the bottleneck link capacity. To gauge the degree to which AFQM achieves fair bandwidth allocation, the average aggregated class throughput in the simulations above, along with their ideal fair shares, are plotted in Fig. 8 where the fraction of total bandwidth versus the UDP flow arrival rate is plotted.

Table 2. Aggregated Class Throughput at the Congested Link: 10 TCP flows with Variable rate UDP flows

Unit: [Mbps]

UDP (Mbps)	TCP (FRED)	UDP (FRED)	Total (FRED)	TCP (RED)	UDP (RED)	Total (RED)	TCP (AFQM)	UDP (AFQM)	Total (AFQM)
1	12.32	1.00	13.31	12.54	0.99	13.53	12.74	0.72	13.46
2	11.45	1.80	13.25	11.12	1.97	13.09	12.72	0.88	13.60
3	10.98	2.28	13.25	10.03	2.93	12.96	12.88	0.89	13.76
4	10.70	2.54	13.24	8.66	3.88	12.54	12.64	0.92	13.56
5	10.60	2.65	13.25	7.50	4.81	12.31	12.57	1.10	13.67
6	10.63	2.65	13.27	6.16	5.72	11.89	12.29	1.28	13.57
7	10.62	2.67	13.29	4.95	6.61	11.56	11.81	1.43	13.23
8	10.57	2.72	13.30	3.86	7.46	11.32	11.87	1.59	13.46
9	10.50	2.78	13.27	2.69	8.23	10.92	11.52	1.91	13.43
10	10.46	2.81	13.26	1.90	8.96	10.87	11.72	2.01	13.73

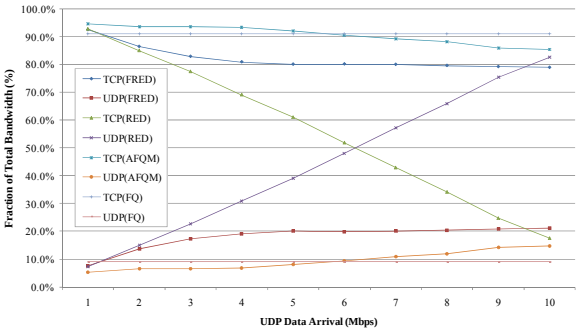

Fig. 8. Fairness comparison between the RED with AFQM, FQ, RED and FRED (10 TCP, 1UDP with variable rates)

Moreover, from the simulation results in Table II, it shows that the packet loss rate in a network under RED with AFQM is small since the TCP flow is responsive to congestion indications and adjusts its packet injection rates accordingly. However, the packet loss rates in a network under RED are large because the RED scheme made unnecessary packet drops due to fault operation. Fault operation occurs when AQM, such as the RED, does not discriminate against unresponsive flows.

4.2 Fragile Flows vs. Robust Flows

As known, if a mixture of short and long RTT flows shares a bottleneck link, then severe unfairness is likely to occur as the short RTT flows grab the available bandwidth well before the long RTT flows have a chance. To evaluate the fairness performance of AFQM by comparing fragile (long

RTT) flows with robust (short RTT) flows, we simulated the configuration shown in Fig. 9.

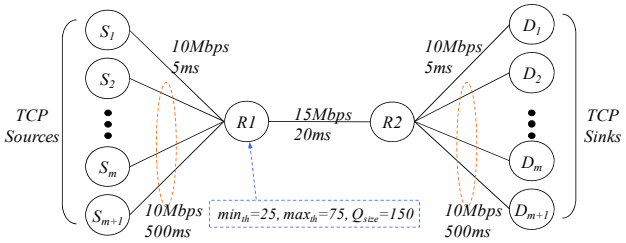

Fig. 9. Network configuration to evaluate the fairness performance of AFQM with fragile connections

A robust TCP connection (flow ID =1) with a RTT of 24 ms and unlimited windows compete with 29 fragile TCP connections with RTT of 1024ms and unlimited windows. Each sender always has data to be sent. The simulation results are summarized in Table 3. Table 3 lists the throughput of each flow in the congested network.

Table 3. Each Flow's Throughput at the Congested Link: A lot of fragile flows and a robust flow

Flow ID	AFQM (Mbps)	RED (Mbps)	Flow ID	AFQM (Mbps)	RED (Mbps)	Flow ID	AFQM (Mbps)	RED (Mbps)
1	*1.1092*	*2.3107*	11	*0.3290*	*0.2099*	21	*0.3206*	*0.2186*
2	*0.3283*	*0.2433*	12	*0.3285*	*0.2547*	22	*0.3147*	*0.2480*
3	*0.3222*	*0.2389*	13	*0.3244*	*0.2436*	23	*0.3106*	*0.2405*
4	*0.3253*	*0.2328*	14	*0.3198*	*0.2694*	24	*0.3307*	*0.2569*
5	*0.3274*	*0.2184*	15	*0.3299*	*0.2353*	25	*0.3280*	*0.2161*
6	*0.3232*	*0.2485*	16	*0.3207*	*0.2303*	26	*0.3292*	*0.2512*
7	*0.3261*	*0.2654*	17	*0.3068*	*0.2559*	27	*0.3315*	*0.2386*
8	*0.3171*	*0.2241*	18	*0.3172*	*0.2637*	28	*0.3328*	*0.2455*
9	*0.3330*	*0.2553*	19	*0.3287*	*0.2423*	29	*0.3327*	*0.2442*
10	*0.3152*	*0.2669*	20	*0.3328*	*0.2600*	30	*0.3163*	*0.2736*
						Total	*10.5119*	*9.4026*

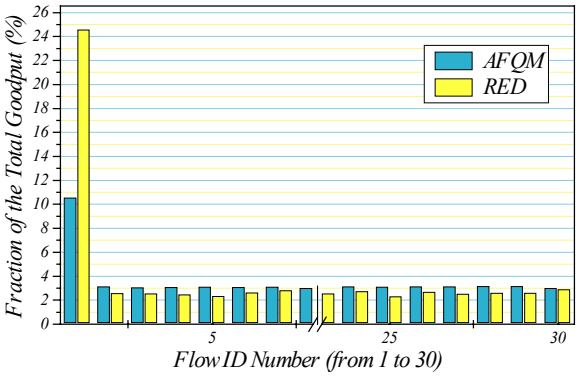

Fig. 10. Fraction of the total Goodput per Flow ID: A lot of fragile flows and a robust flow

From Table 3, the robust TCP (flow ID =1) connection is able to run at the maximum possible speed as if there were no competition. Table 3 shows the actual throughput of the fragile TCP connection and one of the robust TCP connections. The low throughput enables the fragile TCP connection to ramp up to its maximum possible rate. Consequently, the robust TCP flow traffic can take up a large percentage of the link bandwidth and starve out other TCP friendly flows. In order to explore the feasibility of the RED with AFQM, the throughput of each flow is illustrated in Fig. 10. When the achievement of fair bandwidth allocation under RED with AFQM is compared, it is slightly better than that under RED, as shown in Fig. 10.

4.3 Between the transport layer protocols

The most important factor influencing the performance of router congestion control is the transport layer protocols such as TCP-Reno, TCP-Vegas and TFRC (TCP-Friendly Rate Control); the TCP-Reno is the most popular TCP, the TCP-Vegas is a new version of TCP to improve performance and the TFRC is an equation-based congestion control [19-22]. However, the transport layer protocol schemes have unfairness problems between them. To evaluate the fairness performance of the AFQM comparing with that of the transport layer protocols, we simulated the configuration shown in Fig. 11.

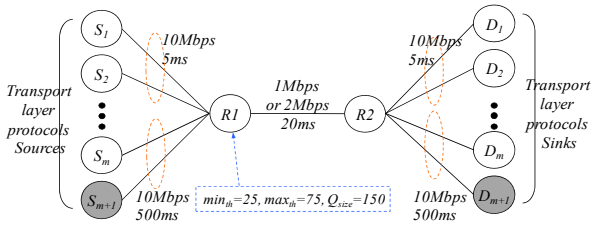

Fig. 11. Network configuration to evaluate the fairness performance of AFQM with the transport layer protocols

The simulations are performed to compare the three transport layer protocols, which are TCP-Reno, TCP-Vegas and TFRC, with the RED with AFQM. Moreover, in order to set 100% of the bandwidth and delay product, one unresponsive flow, which has a high throughput, is sent to the bottleneck link. Table 4 presents the throughput of each transport layer protocol for the different RTTs.

Table 4. Each Flow's Average Throughput at Receiver (BW = 1Mbps): TCP-Reno, TCP-Vegas and TFRC, for short and long RTT

	TCP-Reno Short	TCP-Vegas Short	TFRC Short	Utiliza-tion (%)	TCP-Reno Long	TCP-Vegas Long	TFRC Long	Utiliza-tion (%)
Load	*Unresponsive flow: Target rate to 1.5 Mbps*							
AFQM (Kbps)	211	187	338	***73.7%***	54	44	185	***28.3%***
RED (Kbps)	65	106	217	***38.9%***	16	8	25	***4.9%***
Drop-Tail (Kbps)	114	62	244	***42.0%***	1	15	30	***4.6%***

As Table 4, with the unresponsive flow, the three transport layer protocols show comparably lower performance because the unresponsive flow does not reduce transmission load for congestion control. Table 4

shows the unfairness problems between them while the link utilization shows low usage with Drop Tail and RED. As shown in Table 4, Fault operation occurs when RED does not discriminate against unresponsive flows, but the RED with AFQM improves fairness and link utilization more than that by the simple RED.

Fig. 12 shows the results of Drop Tail, RED and RED with AFQM. The x axis shows the simulation time, and the y axis shows the throughput of each flow that is buffered in a router queue. As shown in Figs. 12 (a) and (b), each flow's throughput is not normalized by the flow fair share between them, and the unresponsive flow grabs the bandwidth. However, from Fig. 12 (c), it is observed that the flow with the RED with AFQM is more normalized by the fair share, overall. The unresponsive flow cannot grab the router bandwidth because it is penalized by the fairness adjustment parameters.

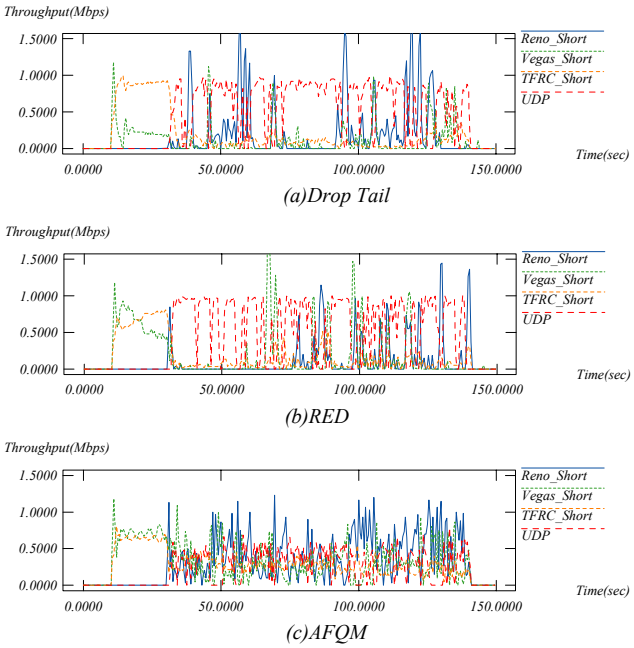

Fig. 12. Each flow's throughput buffered in the router queue: (a) Drop tail, (b) RED, (c) AFQM

In the next simulation scenario, the simulation is setup with different link delays to make short RTT and long RTT flows at the same time. Each of the transport layer protocols sends two flows that are the short RTT and long RTT flows. The simulation results are summarized in Table 5.

It is observable from Table 5 that the usage utilization by the RED with AFQM is much higher than those by others. Also, the fairness of bandwidth sharing among flows using the RED with AFQM outperformed those of the others. The simulation result shows that the AFQM approximates fairness among non-TCP flows or misbehaving flows by increasing and decreasing the drop probabilities in the manner explained in Section 3.

Table 5. Each Flow's Average Throughput at Receiver (BW=2Mbps): TCP-Reno, TCP-Vegas and TFRC, with short and long RTT at Once

	AFQM (Kbps)	RED (Kbps)	Drop-Tail (Kbps)	AFQM (Kbps)	RED (Kbps)	Drop-Tail (Kbps)
Unresponsive flow	*Target rate to 1.5 Mbps*			*Target rate to 0 Mbps*		
TCP-Reno Short	450	310	350	500	550	550
TCP-Reno Long	30	10	10	70	20	20
TCP-Vegas Short	400	410	400	400	750	310
TCP-Vegas Long	30	10	10	60	30	20
TFRC Short	610	310	500	750	680	1180
TFRC Long	30	20	10	260	40	50
Utilization (%)	***77.50%***	***53.50%***	***63.50%***	***100%***	***100%***	***100%***

6. Conclusion

In this paper, the new active queue management algorithm, called the AFQM algorithm, is proposed to improve the fairness of existing AQM algorithms. AFQM aims to approximate the fair queueing policy such as the FQ. In addition, the RED with AFQM scheme can be easily implemented with small overhead. The AFQM scheme provides selective dropping by the fairness adjustment parameters based on per-active-flow buffer counts. The simulations suggest that the RED with AFQM scheme works well in protecting TCP friendly flows from non-TCP flows or misbehaving flows. The simulation results show that RED with AFQM outperforms the RED in terms of fairly handling connections with different RTT and traffic classes. Further work involves studying the proposed algorithm performance under a wider range of parameters, network topologies and real traffic traces, and future work will consider methods for improving fairness in other QoS services.

Acknowledgments

This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology(2011-0012561).

References

[1] Jahon Koo and Kwangsue Chung, "On Improving the Fairness of Active Queue Management for Congestion Control," ICOIN2010, Session 4B-3, January 2010.

[2] Shao Liu, Tamer Basar, and R. Srikant, "Exponential RED: A Stabilizing AQM Scheme for Low- and High-speed TCP Protocols," IEEE/ACM ToN, October 2005.

[3] Stevens, W., "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms," IETF RFC 2001, January 1997.

[4] Floyd, S., and Fall, K., "Router Mechanisms to Support End-to-End Congestion Control," LBL Technical report, February 1997.

[5] Demers, A., Keshav, S. and Shenker, S., "Analysis and Simulation of a Fair Queueing Algorithm," Journal of Internetworking Research and Experience, October 1990. Also in Proceedings of ACM SIGCOMM 89.

[6] A. Legout, and E. W. Biersack, "Revisiting the Fair Queueing Paradigm for End-to-End Congestion Control," IEEE Network Magazine, September 2002.

[7] Stoica, I., Shemker, S., and Zhang, H., "Core-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocations in High Speed Networks," Proceedings of ACM SIGCOMM 98, August 1998.

[8] McKenny, P., "Stochastic Fairness Queueing," Proceedings of INFOCOM 90, pp733-740, June, 1990.

[9] Floyd, S., and Jacobson, V., "Random Early Detection Gateways for Congestion Avoidance," IEEE/ACM Transaction on Networking, August 1993.

[10] Shin, M., Chong, S., and Rhee, I., "Dual-Resource TCP/AQM for Processing-Constrained Networks," IEEE/ACM Transaction on Networking, April 2008.

[11] Tinnakornsrisuphap, P. and J. La, R., "Asymptotic Behavior of Heterogeneous TCP Flows and RED Gateway," IEEE/ACM Transactions on Networking, February 2006.

[12] Padhy, J., Firoiu, V., Towsley, D., and Kurose, J., "A Stochastic Model of TCP Reno Congestion Avoidance and Control," Technical Report CMPSCI TR 99-02, Univ. of Massachusetts, Amherst, 1999.

[13] Lin, D. and Morris, R., "Dynamics of random early detection," ACM SIGCOMM 97, pp 127-137, October 1997.

[14] Ott, T., Lakshman, T. and Wong, L., "SRED: Stabilized RED," INFOCOM 99, March 1999.

[15] Pan, R., Prabhakar, B., and Psounis, K., "CHOKe, A Stateless Active Queue Management Scheme for Approximating Fair Bandwidth Allocation," INFOCOM 2000, February 2000.

[16] X. Yang, H. Chen, and P. Xiao, "An algorithm of enhancing RED fairness," in Proc. of WCICA 2008, June 2008.

[17] Mahajan, R. and Floyd, S. "Controlling High-Bandwidth Flows at the Congested Router," ACIRI, Berkeley, California, November 2000.

[18] UCB/LBNL/VINT, "Network Simulator - ns (Version 2.31)," http://www.isi.edu/nsnam/ns/.

[19] Brakmo, L., and Peterson, L., "TCP Vegas: New Techniques for Congestion Detection and Avoidance," SIGCOMM94, August 1994.

[20] Henderson, T., and Sahouria, E., "On Improving the Fairness of TCP Congestion Avoidance," Proceeding of GLOBECOM1998, July 1998.

[21] Mo, J., Anantharam, R. and Walrand, J., "Analysis and Comparison of TCP Reno and TCP Vegas," GLOBECOM99, December 1999.

[22] Floyd, S., Handley, M., Padhye, J., and Widmer, J., "Equation-based congestion control for unicast applications," SIGCOMM2000, August 2000.

Jungmin Lee received the B.S. degree and M.S. degree from Kwangwoon University, Seoul, Korea, all from the Electronics & Communications Department, in 2000 and 2002 respectively. Currently he is pursuing Ph.D. degree in Communications Engineering Department at Kwangwoon University. His research interests include multimedia streaming, congestion control, mobile IPTV, and Internet QoS.

Kwangsue Chung received the B.S. degree from Hanyang University, Seoul, Korea, M.S. degree from KAIST (Korea Advanced Institute of Science and Technology), Seoul, Korea, Ph.D. degree from University of Florida, Gainesville, Florida, USA, all from Electrical Engineering Department. Before joining the Kwangwoon University in 1993, he spent 10 years with ETRI (Electronics and Telecommunications Research Institute) as a research staff. He was also an adjunct professor of KAIST from 1991 to 1992 and a visiting scholar at the University of California, Irvine from 2003 to 2004. His research interests include communication protocols and networks, QoS mechanism, and video streaming.