Adaptive Queue Management with Restraint on Non-Responsive Flows

Lan Li and Gyungho Lee
Department of Electrical and Computer Engineering
University of Illinois at Chicago
851 S. Morgan Street Chicago, IL 60607
{lli, ghlee}@ece.uic.edu

ABSTRACT

This paper proposes an adaptive queue management scheme (adaptive RED) to improve Random Early Detection (RED) on restraining non-responsive flows. Due to a lack of flow control mechanism, non-responsive flows can starve responsive flows for buffer and bandwidth at the gateway. In order to solve the disproportionate resource problem, RED framework is modified in this way: on detecting when the non-responsive flows starve the queue, packet-drop intensity (Max_p in RED) can be adaptively adjusted to curb non-responsive flows for resource fair-sharing, such as buffer and bandwidth fair-sharing. Based on detection of traffic behaviors, intentionally restraining non-responsive flows is to increase the throughput and decrease the drop rate of responsive flows. Our experimental results based on adaptive RED shows that the enhancement of responsive traffic and the better sharing of buffer and bandwidth can be achieved under a variety of traffic scenarios.

Keywords
Active queue management, drop probability, non-responsive flows, adaptive RED.

1. INTRODUCTION

Congestion control heterogeneity in the Internet leads to fairness problems of resource sharing. Especially, non-responsive flows do not back off their transmission rates in response to congestion indications of the network (i.e. packet loss) [5]. Consequently, non-responsive flows tend to consume unfairly resource—high bandwidth and buffer space [6] at congestion points. A large number of UDP-based applications generate this kind of flows. It is necessary to employ a flow management mechanism for the fair utilization of network resource. It may also be helpful to satisfy more users' QOS requirement.

There are two kinds of flow management mechanisms that try to achieve the resource fair sharing: scheduling scheme and queue management scheme. Scheduling schemes have generally too much complexity and low scalability to large number of flows. If we plan to provide fair utilization on high-speed edge router, queue management scheme could be a better choice. Queue management scheme not only has less complexity, but also approximates fairness better. Router with a queue management scheme maintains a single FIFO shared by all flows [1] and employs a dropping algorithm to discard arriving packets when congestion builds up. Dropping probability increases with the raise of congestion level. By adjusting dropping probability for in-queue flows, it can achieve a better fairness for the resource utilization between flows.

One of classic queue management schemes is Random Early Detection (RED). The RED utilizes a random dropping algorithm [4]. In a RED gateway, incipient congestions are detected by estimated average queue size. Average queue size is calculated via a low-pass filter. When the average queue size exceeds a preset threshold, the gateway drops packets with a certain probability that is calculated based on the average queue size. In the RED gateway, occasional bursts of packets in the queue are allowed and average queue size is kept at low level [6]. Although RED decreases loss-bias against responsive traffic that exists in the Drop Tail queue, it does not achieve a fair-share of the buffer and bandwidth between responsive flows and non-responsive flows [2]. One possible reason is that given a certain time slot the percentage of packets dropped from each flow is almost the same. As a result, non-responsive flows may consume a larger portion of the bandwidth at congestion points because of its lack of flow control mechanism [3]. Consequently, non-responsive flows can starve out the responsive flows.

In order to solve the disproportionate resource problem, several variants of RED have been proposed, such as Flow Random Early Drop (FRED) [7], Refined Design of Random Early Detection Gateways [6] and Self-Configuring RED Gateway [9]. However, FRED maintains extra flow state which increases implementation overhead. As for the Refined Design of RED, it dynamically adjusts the Wq (queue weight) and Max_p (maximum drop probability) with respect to the variance of queue size change. However, it divides the area between minimal threshold and maximal threshold, called Yellow area [4], into several sub-phrases that increases implementation overhead as well. In addition, it still does not consider the fair sharing of the buffer and bandwidth between responsive flows and non-responsive flows. Self-Configuring RED Gateway also employs an adaptive parameterization at RED gateway. Without considering non-responsive traffic, adaptive parameterization is only based on the congestion level indicated by the average queue length. It can effectively adapt to traffic scenarios with different congestion levels. However, it does not distinguish responsive and non-responsive traffic. In some cases of medium congestion, responsive traffic may get suffered due to the unfriendly behaviors of non-responsive traffic.

If we can distinguish non-responsive flows from responsive flows, a resource fair-sharing between them can be properly managed. Unfortunately, any methods based on per-flow state must increase the implementation overhead. However, we are aware of the approaches that try to approximately discern non-responsive flow and responsive flow according to certain traffic features [5]. Such a feature could be an extraordinary high congestion or an improper reaction to congestion notification.

We then propose to detect flow's behavior based on the variance of instantaneous queue size. Based on the detection, we can penalize non-responsive flows and improves responsive flows' throughput via adaptive adjustment of the drop probability.

The rest of the paper is organized as follows. Section 2 presents flow behaviors in conjunction with the RED queue and notes some flaws of RED algorithm. Section 3, describes our proposed queue management scheme "Adaptive RED with restraint on non-responsive flows". In Section 4, we compare our approach with other schemes via simulation. Finally, we give discusses and conclusions in Section 5.

2. BACKGROUND

2.1 RED Scheme Overview

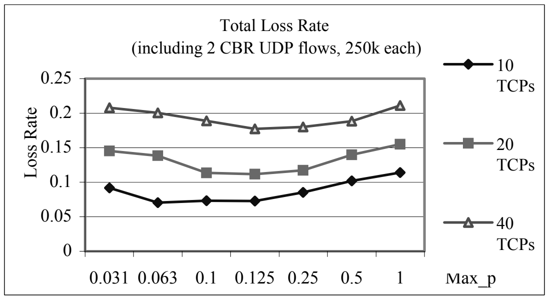
RED (Random Early Detection) is a congestion avoidance mechanism employed at gateways. Its basic idea is to use two preset thresholds to detect incipient congestion and control the average queue size with the threshold [2]. With respect to the estimated average queue size, the gateway works in one of three working states: red, green and yellow states [6]. When the average queue size is less than the Min_th (minimum threshold), the gateway works in the green state without packet drop. When the length is between the Min_th and Max_th (maximum threshold), the state transfers to yellow one in which arriving packets are randomly dropped with a probability calculated on average queue length. When the length goes beyond the maximum threshold, the gateway reaches the red state in which every arriving packet is discarded [6]. The goal of RED scheme is to detect incipient congestion via average queue size and perform random packet dropping before the queue is full. Therefore, it can implement congestion avoidance while allowing short-term bursty.

Although RED prevents packets from consecutive dropping (especially for bursty traffic) and removes higher loss bias against bursty traffic [4], it still has some flaws:
1. RED is unable to refrain non-responsive flows from consuming most buffer space at heavy congestion [3].
2. Average queue size is not a good estimator of congestion. Severe congestion should be related to the amount of flows and the degree of burstiness [2].
3. It is not clear how to select the RED parameters. RED requires selecting various parameters in different congestion scenarios. That is why recommended values had been changed over time [2].

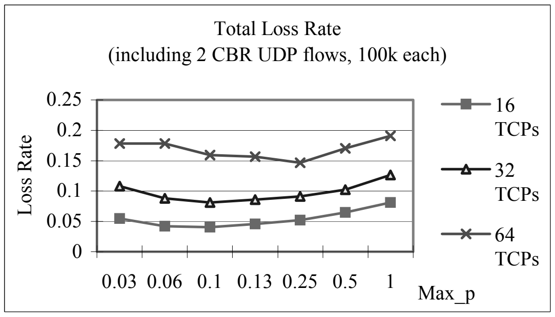
2.2 Flow Behaviors in the RED Queue

Generally speaking, most of TCP flows with congestion control are regarded as responsive flows, while UDP flows without congestion response are considered non-responsive flows [5]. TCP congestion control utilizes an *Additive Increase/Multiplicative Decrease* mechanism to adjust the flow rate [8]. This mechanism causes the burstiness of TCP flows. Since TCP congestion control is based on end-to-end feedback, the control response is relatively sluggish in comparison with queue operation. It implies that TCP would keep the sending rate for a while after packet dropping. The control latency is equal to the TCP timeout, which is usually four times as much as RRT (Round Trip Time) [8]. On the other hand, non-responsive flows have no congestion control mechanism. They do not respond to congestion notification and thus send packets as many as they want.

At the RED-suite gateways, TCP flows (responsive flow) tend to decrease sending rate after average queue size goes beyond the minimal threshold. However, RED does not penalize non-responsive flow, because packet drop ratio of each flow over time is almost same. Although UDP flows (non-responsive flow) with high bandwidth may suffer more packet dropping, they obtain much more throughput than TCP flows do. Consequently, non-responsive flows may take up the most bandwidth and starve out responsive flows [3]. Since RED uses static parameters to deal with dynamic traffic, the effectiveness of RED depends on the appropriate parameterization. It is known that no single set of RED parameters works well under different congestion scenarios, including the case that TCP is mixed with UDP.



(a) Two 250k UDP flows are mixed with TCP flows



(b) Two 100k UDP flows are mixed with TCP flows

Figure 1. TCP loss rate vs. Max_p

Simulation result in Figure 1 shows that the total loss rate is pretty high for the small Max_p. As Max_p increases, loss rate decreases since the RED queue is able to send congestion notification back to the sources in time to prevent continuous buffer overflow. Finally, when Max_p becomes quite large, the RED queue causes an increase in packet loss rates over Drop Tail queues. As more connections are added, the optimal value of Max_p increases. It clearly supports that an adaptive scheme for drop probability is required. More extensive and detailed discussion can be also found in [9]. The details of the simulation are fully described in Section 4.

3. ADAPTIVE RED WITH RESTRAINT ON NON-RESPONSIVE FLOWS

Though TCP flows are dominant traffic in Internet, more and more multimedia applications tend to generate UDP flows, mostly belonging to non-responsive flows. It is because those applications mainly concern the delay constraint instead of data integrity. It is necessary to deploy a flow management mechanism to implement a fair utilization of network resource.

3.1 Basic assumption

Given that packet-processing rate is invariable (service rate: S_r) and non-responsive flows have the arrival rate $R_{non}(t)$, the instant queue size is determined by:

$$Q_{inst}(t) = \int_{t_i}^{t} (R_{non}(t) + R_{rep}(t) - S_r)\,dt \qquad (1)$$

, where $R_{rep}(t)$ is the arrival rate of responsive flows; t_i is the initialization time.

The average rate of n TCP-friendly flows (responsive flow) can be approximately calculated as [10]:

$$r_{rep} = \frac{n}{RTT}\sqrt{\frac{3}{2p}}\ \text{packets}/\text{second} \qquad (2)$$

, where RTT is the round trip time and p is the loss probability of flow. It is applicable only when the p is small. According to the Eq. (1), given a small time slot T with the size of Δt , change in queue size is mainly determined by the arrival rate of flows:

$$\frac{\Delta Q_{inst}(\tau)}{\Delta t} \cong R_{non}(\tau) + R_{rep}(\tau) - S_r$$

Considering two consecutive slots T_j and T_{j+1} with the same size Δt , the variance of arrival rate Δr ($\Delta r = \Delta r_{non} + \Delta r_{rep}$) can be approximated by:

$$\Delta r \cong \frac{\Delta Q_{inst}(\tau_{j+1}) - \Delta Q_{inst}(\tau_j)}{\Delta t^2}$$

, where $\tau_j \in T_j$ and $\tau_{j+1} \in T_{j+1}$ (3)

Suppose r_{rep}^{j} is the average arrival rates of responsive flows at slot T_j and $\Delta t \approx RTT$, extending Eq. (2) the variance of responsive flows' arrival rate

$$\Delta r_{rep} \cong \left(\sqrt{\frac{p_j}{p_{j+1}}} - 1\right) r_{rep}^{j} \qquad (4)$$

, where p_j and p_{j+1} are drop probability at T_j and T_{j+1} .

3.2 RED Queue Analysis

The very important feature of RED algorithm is the use of a low-pass filter to estimate the average queue size [6]. The low-pass filter employs the EWMA (exponentially-weighted moving average) given by [4]:

$$Q_{ave} = (1 - w_q)Q_{ave}' + w_q Q_{inst}$$

,where w_q is the weight factor. Q_{ave} and Q_{ave}' are the current and previous average queue size respectively.

Figure 2 shows the drop function of RED. Drop rate of RED is presented by:

$$D_{red} = \begin{cases} 0 , & Q_{ave} < Min\_th \\ \dfrac{Q_{ave} - Min\_th}{Max\_th - Min\_th} Max\_p, & Min\_th \le Q_{ave} < Max\_th \\ 1 , & Max\_th \le Q_{ave} \end{cases} \qquad (5)$$

Extending Eq. (4), we can have:

$$\Delta r_{rep} \cong \left(\sqrt{\frac{Q_{ave}(\tau_j) - Min\_th}{Q_{ave}(\tau_{j+1}) - Min\_th}} - 1\right) r_{rep}^{j} \qquad (6)$$

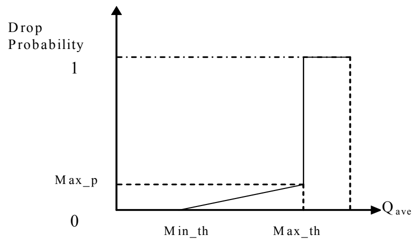


Figure 2. Drop function of RED

The tendency of responsive flows can be detected via $\Delta Q_{ave} (\Delta Q_{ave} = \Delta Q_{ave}(\tau_{j+1}) - \Delta Q_{ave}(\tau_j))$. When Q_{ave} increases to the "yellow state" [2], r_{rep} should decease accordingly. However, if r_{non} gets inverse tendency at that time, the arrival rate of all flows (r) increases. It causes responsive flows starved and resource unfairly shared.

3.3 Adaptive RED Scheme

Adaptive RED is proposed to detect non-responsive (unfriendly) flow behaviors via monitoring the variation of queue length, including instant and average queue length. At congestion, responsive flows decrease its rate while non-responsive flows may keep or increase the rate. Therefore, adaptively adjusting packet-drop intensity may restrain non-responsive flows to starve responsive flows. We apply this adaptive scheme only if the average queue length ranging from Mid_th ((Min_th+Max_th)/2) to Max_th. In that medium congestion area, responsive flows just get essential suffering and take deep congestion response.

Based on our assumption and analysis in Section 3.1 and 3.2, ΔQ_{ave} indicates the tendency of responsive flows and $\Delta^2 Q_{ins}$ ($\Delta^2 Q_{inst} = \Delta Q_{inst}(\tau_{j+1}) - \Delta Q_{inst}(\tau_j)$) reflects the variation of arrival rate Δr . If both of them are positive, Δr_{non} must be positive too. It implies non-responsive flows take unfriendly response against responsive flows and the ratio of non-responsive flows and responsive flows gets increased. At that time, unfair resource sharing happens and the adjustment of drop density is needed. In order to avoid too aggressive of packet dropping, drop density needs to be adjusted back once the condition is not satisfied ($\Delta^2 Q_{inst} < 0 \mid \Delta Q_{ave} < 0$). We then propose our algorithm as follows:

```
Every T (control slot):
While ( Max_th > Q_ave > Mid_th)
    if $\Delta^2 Q_{inst} > 0$ & $\Delta Q_{ave} > 0$
        Max_p = Max_p*α
    else
        Max_p =Min (Max_p0 , Max_p/α)
Max_p0: Default Max_p
α : the scale factor of Max_p
```

α is the scale factor to extend the Max_p. Max_p0 is the preset parameter of drop density. Based on the observation of ΔQ_{ave} and $\Delta^2 Q_{inst}$, the algorithm detects unfriendly traffic behaviors and modulates the value of Max_p by the factor of α. The parameters α is experimentally chosen according to the control slot (T) which determines the control sensitivity. If control slot is too small, there is no enough time for responsive flows to response and the scheme may become more aggressive and cause control oscillation. On the other hand, it may become insensitive and cause control sluggishness. In the simulation, we chose control time slot as 0.2s (average RTT). This slot size can accommodate the short burstiness and avoid the control vibration.

4. SIMULATION RESULTS

This section presents the simulation results of Adaptive RED's performance with refraining of non-responsive flows and enhancing the responsive flows. The simulations range over network configurations and traffic patterns.

4.1 Basic network Configuration

Network simulation topology is shown in Figure 3. There is a single congested link from GW1 to GW2 in a dumbbell topology. The congested link is 1Mbps; others are 10 Mbps each. TCP and UDP flows pass through the congested link. The number of TCP flows ranges from 8 to 32, while that of UDP flows ranges from 2 to 8. Three active queue management schemes: RED, Self-Configure and ARED are applied on the bottleneck link.

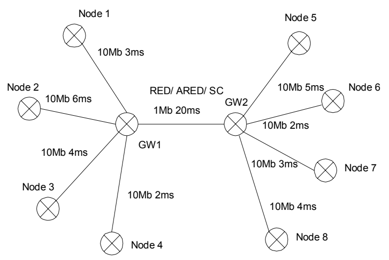


Figure 3. Topology of the Simulation

To compare with RED and Self-Config schemes, a popular simulation environment of RED-suite gateway is chosen and described in Table 1 and Table 2. The physical queue capacity is set to 300 packets. Following [4], we set Min_th to 100 packets, and set Max_th to be twice as the Min_th. RED is simulated with NS's default value of W_q=0.002 and Max_p=0.1. Parameters for Self-Config are referred to [6] in which α =3 and β =2. We experimentally select T as 0.2s and α as 1.5 to ARED. In order to create non-responsive behaviors, UDP flows for all schemes are chosen as exponential type with a packet size of 210 bytes.

Queue Capacity	300 Packets
Min_th	100 Packets
Max_th	200 Packets
α for ARED	1.5
T (Control slot)	0.2 s
W_q and Max_p for RED	0.002 and 0.1
α β for Self-Config	3 and 2

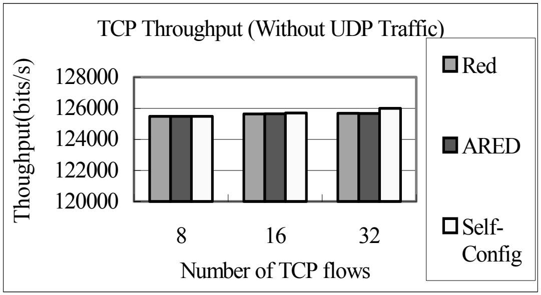
Table 1. Gateway Parameters

TCP window size	15
UDP packet-size	210
TCP packet-size	1000
UDP Type	Exponential
TCP Type	FTP traffic
UDP rate	0.25 Mbps

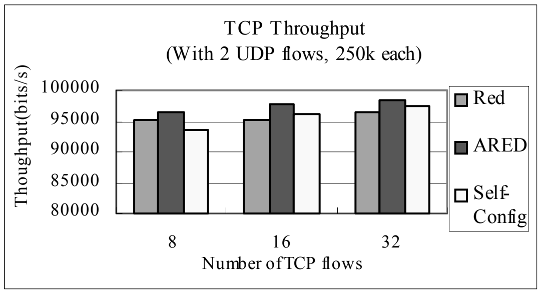
Table 2. Flow Parameters

4.2 Simulation Results

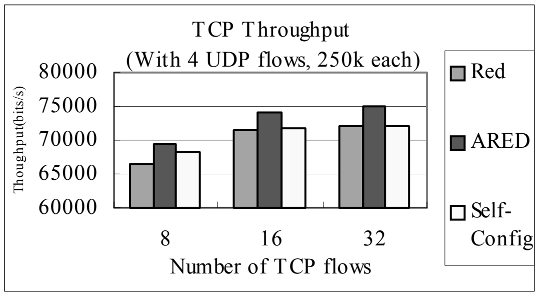
1) The throughput of RED, Self-Config and Adaptive RED respectively



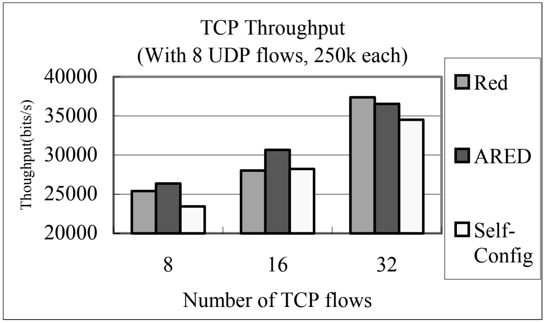
(a) Pure TCP flows



(b) Mixing with 2 UDP flows



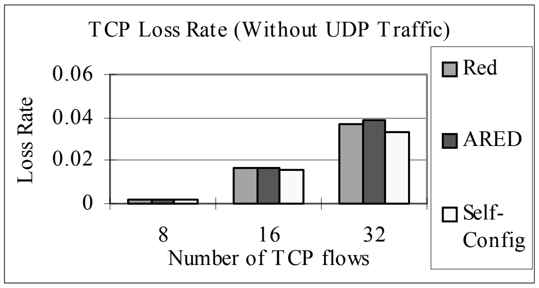
(c) Mixing with 4 UDP flows



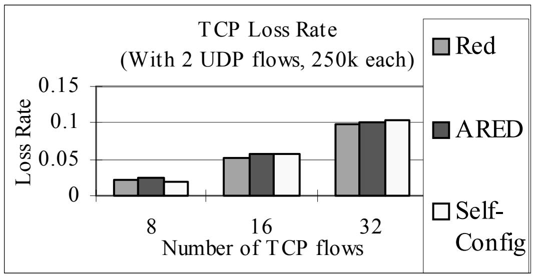
(d) Mixing with 8 UDP flows

Figure 4. TCP Throughput Performance

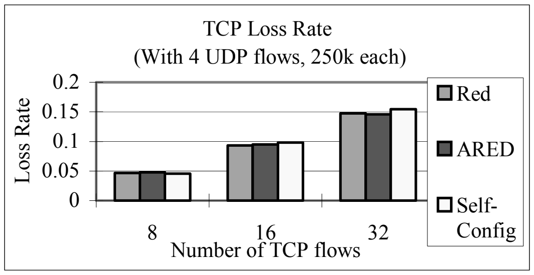
2) TCP Loss Rate of three schemes



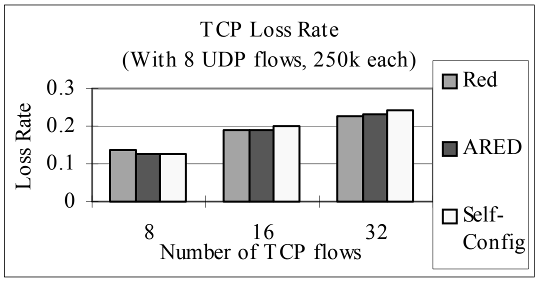
(a) Pure TCP flows



(b) Mixing with 2 UDP flows



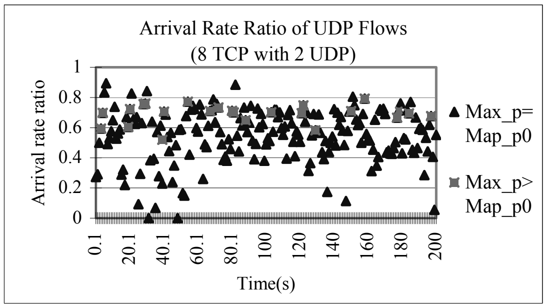
(c) Mixing with 4 UDP flows



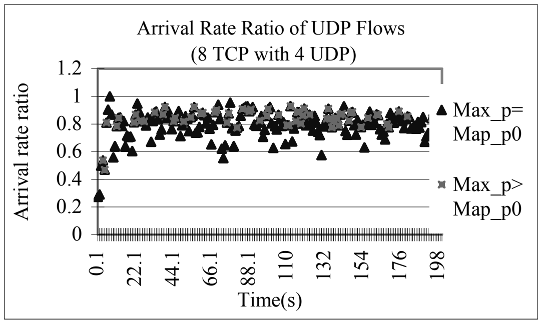
(d) Mixing with 8 UDP flows

Figure 5. TCP Loss Rate

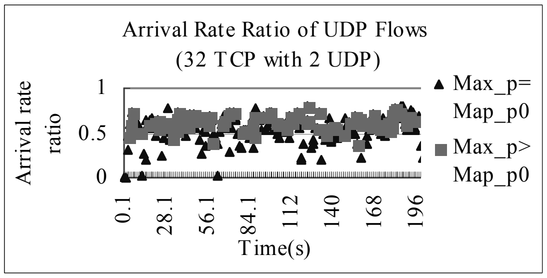
3) Restraint on unfriendly non-responsive traffic by ARED.



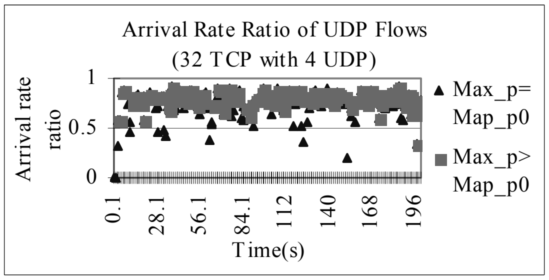
(a) 8 TCP flows mixing with 2 UDP flows



(b) 8 TCP flows mixing with 4 UDP flows



(c) 32 TCP flows mixing with 2 UDP flows



(d) 32 TCP flows mixing with 4 UDP flows

Figure 6. Arrival Rate Ratio of UDP flows

TCP throughput is shown in Figure 4. Compared with RED and Self-Config, ARED scheme has comprehend-sively better performance. In case of pure TCP (Figure 4(a)), Self-Config scheme shows slightly better. It is because Self-Config adjusts drop density only according to the congestion level. To do so can help avoid heavy and short congestion caused by a large number of TCP flows. When a large number of TCP flows mix with big UDP traffic (Figure 4(d)), Self-Config scheme becomes the worst one. According to the algorithm, Max_p of Self-Config may always be kept in a big value, if average queue length fluctuates around Max_th [9]. Such adjustment to Max_p may be so aggressive that TCP suffers more than UDP, since UDP flows always overwhelm TCP flows so that the bandwidth released from TCP is taken over by UDP.

Corresponding TCP loss rate is shown in Figure 5. Basically, the decease of TCP drop rate indicates the improvement of TCP throughput. It implies that ARED helps TCP flows scramble for bandwidth from UDP flows. It makes responsive flows be enhanced. However, it can be found that a little bit lower performance than RED is presented when a large number of TCP and UDP flows are mixed (32 TCP flows in Figure 5(d)). The reason may be that with severe and continuous congestion, ARED has less space to operate so that it may misdetect non-responsive behaviors. The similar problem would be found in the simulations of more UDP flows. However, there is no need to discuss those cases because of very severe congestion and intolerable drop ratio (beyond 30%). Those cases should be in system problem and cannot be solved just by queue management schemes.

Figure 6 shows the effectiveness of ARED. We sampled the arrival rate ratio of UDP flows ($\frac{r_{non}}{r}$) every 0.5s. Most drop density adjustments are made when UDP flows occupy more than half of the bandwidth. At that time, more UDP packets are dropped via increasing Max_p. In addition, ARED adjusts Max_p only when the average queue length goes beyond Mid_th. Therefore, ARED effectively refrains non-responsive traffic behaviors and enhances responsive ones as well. We would not list the simulation result of Mixed traffic with 6 and 8 UDP flows, because the result does not mean much if UDP flows always overwhelm TCP flows.

As a conclusion, ARED keeps responsive flows more adaptively than RED and Self-Config do. It also has better throughput performance and can adapt to a wide variety of traffic scenarios.

5. DISCUSSIONS AND CONCLUSION

We only applied ARED to exponential UDP traffic in our simulation because of its simplicity and typical non-responsive feature. With regard to the basis of ARED scheme, any unfriendly non-responsive traffic can be detected and refrained by ARED, such as DDOS attack flows. Though DDOS attack may generate TCP flows, the congestion mechanism does not work. The attack traffic shows absolutely non-responsive behaviors. ARED has great chance to detect and restrain it. We can also adaptively adjust parameter of α so that ARED can fit well with the different degree of non-responsive behaviors. It is expected that α is revised according to the difference between Δr and Δr_{rep}.

This paper has shown how adaptive active queue management can be used in conjunction with RED gateway to effectively enhance responsive behaviors. Preliminary simulation results with a popular environment have shown the efficacy of the algorithm. If the interactions among different flows are fully understood and control parameters are well optimized, more performance improvement can be expected in an extensive research. Since the networks are becoming more heterogeneous, this scheme should adapt to more extensive traffic. How to improve the scalability is our future work.

6. REFERENCES

[1] W. Feng, K. G. Shin, D. Kandlur, and D. Saha, "Stochastic Fair Blue: A Queue Management Algorithm for Enforcing Fairness", **Proc. IEEE INFOCOM**, Anchorage, AK, 2001, 1520-1529.

[2] T. Bonald, M. May and J. Bolot, "Analytic Evaluation of RED Performance", **Proc. IEEE INFOCOM**, Tel Aviv, Israel, 2000, 1415-1424.

[3] R. Pan, B. Prabhakar, K. Psounis, "CHOKE: A Stateless Active Queue Management Scheme for Approximating Fair Bandwidth Allocation", **Proc. IEEE INFOCOM**, Tel Aviv, Israel, 2000, 942-951.

[4] S. Floyd, and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance", **IEEE/ACM Transactions on Networking,** *1*(4), 1993, 397-413.

[5] S. Floyd, and K. Fall, "Promoting the Use of End-to-end Congestion Control in the Internet", **IEEE/ACM Transactions on Networking**, 7(4), 1999, 458-472.

[6] H. Wang and K. G. Shin, "Refined Design of Random Early Detection Gateways", **Proc. IEEE GlobeCom**, Rio de Janeiro, Brazil, 1999, 769-775.

[7] D. Lin and R. Morris, "Dynamics of Random Early Detection", **Proc. ACM SIGCOMM,** Cannes, France, 1997, 127-137.

[8] W. Stevens, **TCP/IP illustrated, Volume 1: the protocols** (Reading, MA: Addison-Wesley, 1994)

[9] W. Feng, D. Kandlur, D. Saha, K. Shin, "A Self-configuring RED Gateway", **Proc. IEEE INFOCOM**, New York, NY, 1999, 1320-1328.

[10] J. Padhye, V. Firoiu, D. Towsley and J. Kurose, "Modeling TCP Throughput: A Simple Model and Its Empirical Validation", **Proc. ACM SIGCOMM**, Vancouver, Canada, 1998, 303-314.