Queue Management of RIO to achieve high throughput and low delay

堀, 良彰
九州芸術工科大学芸術情報設計学科

https://doi.org/10.15017/4060994

出版情報：芸術工学研究. 3, pp.23-29, 2001-02-27. 九州芸術工科大学
バージョン：
権利関係：

Queue Management of RIO to achieve high throughput and low delay

堀　良彰　　池永全志　　尾家祐二
Hori Yoshiaki　　Ikenaga Takeshi　　Oie Yuji

We have focused on the RIO queueing mechanism in statistical bandwidth allocation service, which uses AF-PHB. We have studied the parameterization of RIO to achieve both high throughput and low delay. We were able to parameterize RIO for that purpose in terms of both *min_th* and *maxp* used in dropping "*Out*" packets. Furthermore, we have also examined the parameterization regarding EWMA (Exponential Weighted Moving Average), i.e., weight factor $w_{qout,}$ and have shown that dropping "*Out*" packets should depend upon the queue length without much delay unlike in RED. From our simulation results, we could see that our parameterization provided high throughput performance and also limited the queue length in a narrow range very well.

1. Introduction

The architecture of differentiated services (diffserv)[3] has received much attention recently as a promising architecture for providing a various levels of service. In the current Internet backbone, which serves IP packets as the "Best Effort" service model, the packets are treated equally on routers in principle and are processed in the same queue. Therefore, when an output link is congested and the buffer equipped with it becomes full, packets are dropped equally. However, there are in fact many different requirements on the quality of services today, and the diffserv architecture has therefore been proposed to meet them from a practical point of view.

In the diffserv architecture, IP packets are marked with DiffServ Code Point (DSCP)[2] according to the contract between the Internet Service Provider (ISP) and users before entering the network. Core routers check the DSCP on packets and treat them according to the DSCPs. IETF's diffserv working group has made the related standard. Expedited Forwarding Per Hop Behavior (EF-PHB)[5] and Assured Forwarding PHB Group (AF-PHB)[4] and their DSCPs have already been defined.

For the purposes of this paper, we focus on the RIO (RED[1] with In/Out) queueing mechanism in statistical bandwidth allocation service, which uses AF-PHB. RIO based schemes have been proposed as a simple mechanism to provide statistical bandwidth allocation

for TCP flows. RIO mechanism has two kinds of packets, i.e., *"In"* and *"Out"* packets, which corresponds to two delivery classes, i.e., AF11 and AF12 in Assured Service, respectively. If the packet arrival rate exceeds a predetermined target rate, arriving packets will be marked with *Out*. Otherwise, they will be marked with *In*. When an *"In"* packet reaches the RIO queue, RIO calculates an average number of *"In"* packets and decides whether to enqueue or to drop it depending on the resulting average. *"Out"* packets are managed in a similar way; an average queue of total packets including *"In"* and *"Out"* packets are used. Since *"In"* packets have the preference over *"Out"* packets, *"Out"* packets can be dropped even when the router is lightly loaded. Combining the above marker and dropper mechanisms enables a bandwidth allocation service in the Internet.

In particular, we studied the parameterization of RIO to achieve both high throughput and low delay performance. That can be achieved in a way to avoid long queues as well as to minimize the probability that the buffer becomes empty. An empty buffer results in inefficiency, which in turn leads to low throughput. On the other hand, long queues cause large delays.

This paper is organized into several sections. Section 2 describes the simulation experiments. Section 3 shows the simulation results with respect to characteristics of the RIO queuing. Section 4 concludes the paper and discusses future works.

2. Simulation Model

In this study, we investigated the characteristics of RIO mechanism by computer simulations. We use Network Simulator version 2 (ns-2) [8] offered by VINT Project and Sean Murphy's diffserv package [9] that is contributed package for ns-2. We also add some codes to simulate Time Sliding Window (TSW) tagger and so on.

We used a simple network setup with twenty sources *S1~S20* sent TCP traffic to destinations *D1~D20* via router *G1* and *G2* as shown in Figure 1. *G1* has a bottleneck link to *G2* with RIO output queue.

Each source node was connected to *G1* with 100Mbps link whose delay ranged from 3ms to 8ms. Each destination node was also connected to *G2* with 100Mbps link whose delay was 5ms. *G1* and *G2* were connected with 70Mbps link whose delay is 5ms. So that end-to-end round trip time ranged from 26ms to 36ms. *S1~S20* transmitted TCP traffic to *D1~D20*. Each source node sent one TCP flow. Router *G1* had TSW tagger on each input interface. Each TSW tagger was able to configure a parameter, *Target Rate (Rt)*. RIO queue of *G1* had a buffer of 300 packets in size.

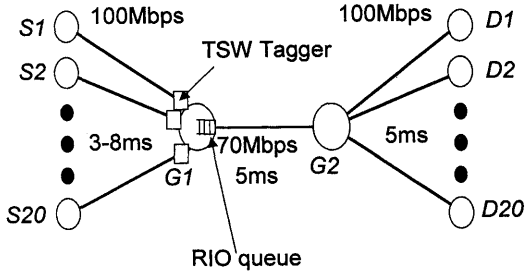

Fig. 1. Simulation model

Each TCP flow conveyed a bulk-data transfer through FTP. TCP segment size was 500bytes. Three TCP variants [11] were treated here: Reno, Newreno, and SACK. Simulation time was twenty seconds. Each source started TCP at *0th* second and traffic behavior became stable in five seconds.

The TSW tagger has a mechanism to avoid continuous *"Out"* marking and RIO queue further has a mechanism to avoid continuous dropping packets. They were proposed by D. Clark and W. Fang [6].

The target rates of ten TSW taggers were set to 5Mbps. The other ten TSW taggers were set to 1Mbps. Therefore the total target rate was 60Mbps and this network because over-provisioned by 10Mbps as shown Fig.1.

3. Simulation Results

In the research, we assumed an over-provisioned network as stated earlier. Because of the excess capacity provided and TCP flow control trying to use all the available bandwidth, packets could be marked with AF12(*Out*) as well as AF11(*IN*). AF11 packets should have priority over AF12 packets due to the features described earlier. Therefore threshold parameters of RIO queue were set, e.g., as follows.

for AF11: (minth_in, maxth_in)

= (200, 250)

for AF12: (minth_out, maxth_out)

= (150, 200)

In this case, as long as the average queue length was under 200 packets, AF11 packets were protected whereas AF12 packets could be dropped. In our simulation, no AF 11 packets were, in fact, dropped because we used the over-provisioned network.

The acceptable status of a buffer is categorized into three phases according to W. Fang et al. [7]. In congestion free phase (phase 1), even no "*Out*" packets are dropped. In congestion tolerance phase (phase 3), all "*Out*" packets are dropped but no "*IN*" packet is dropped. As an intermediate phase, in congestion sensitive phase (phase2), only "*Out*" packets are dropped with some probability, which allows flows exceeding their target rates to increase their window size. Phases 2 and 3 may be considered as desirable ones in that the contract can be kept and queue length is limited within a small range. Although "*Out*" packet indicates excess traffic and thus can be dropped, even only one dropped packet on a flow causes the fast recovery, thereby halving the congestion window size of the flow, i.e., halving the throughput for a while. Furthermore, multiple packet loss occurring in a window can lead to timeout for retransmission in Reno, which stops transmitting packets for a while, resulting in sever throughput degradation. Therefore, phase 3 should be avoided if possible and in phase 2 even a small probability of dropping "*Out*" packets can contribute to limiting the throughput of flow with excess traffic. Relatively high drop probability must provide severe throughput degradation.

From this viewpoint, in what follows, we first determined two parameters associated with AF12, i.e., *minth* and *maxp*, in a way to maximize the throughput as well as to minimize the average queue length. We also focused on the probability of an empty queue.

A. *Effect of minth*

We first investigated the effects of *minth* of AF12. Parameters were set as follows:

For AF11:(minth, maxth, maxp)

= (200, 250, 1/50)

For AF12: (minth, maxth, maxp)

= (*minth_out*, 200, 1/10)

(*minth_out* = 20, 40, 60, 80, 100)

Parameter *minth* varies from 10 to 100. Table. 1 shows the total throughput of TCP flows and the average (ql_ave) and standard deviation (ql_std) of queue length in 5~20 seconds.

Tbl.1 Total throughput and Queue length

Minth_out	TCP	Through-put	ql_ave	ql_std
20	Reno	68.63	26.75	16.29
	Newreno	68.56	27.24	17.03
	Sack	68.51	27.27	17.22
40	Reno	68.73	38.78	22.63
	Newreno	68.98	40.37	23.45
	Sack	69.05	40.93	23.24
60	Reno	68.89	49.72	28.69
	Newreno	69.20	53.20	28.12
	Sack	69.33	54.31	28.61
80	Reno	69.25	61.48	34.62
	Newreno	69.43	66.61	33.71
	Sack	69.44	65.77	32.80
100	Reno	69.19	71.27	39.34
	Newreno	69.45	78.70	37.07
	Sack	69.60	77.87	36.99

From Table 1, we can see that the total throughput performance improves as *minth* increases for all three TCP variants. With *minth* larger than or equal to 60, all the TCP variants achieve almost the same total throughput. On the other hand, the queue length becomes longer with the increase of *minth*. In addition, we see that, for example, with *minth* of 60 packets, the average queue length can be limited very well to approximately 50 packets in all the TCP variants for *maxp* of 1/10. For these reasons, we set *minth* for "*Out*" packets at 60 as an optimum value below.

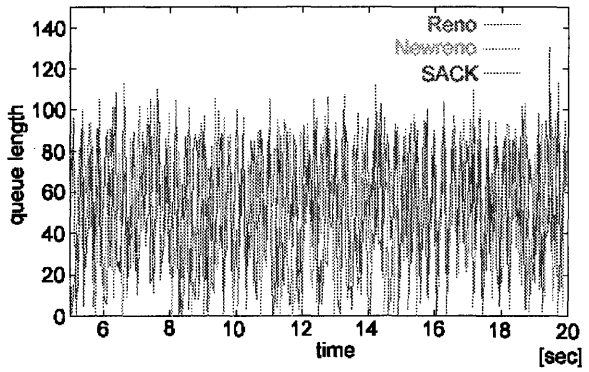

Fig.2 Dynamics of queue length

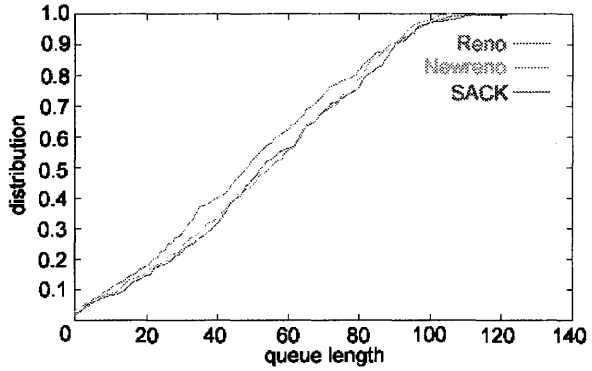

Fig.3 Distribution of queue length

Figure 2 shows how the queue length varies with time and Fig. 3 shows the distribution of queue length when *minth*=60. There is not a significant difference among the distributions of the three TCPs. In all the cases, the buffer is empty with approximately 2%. In addition, we see that the queue length is almost uniformly distributed over a range of less than 80 since the corresponding distribution function increases in a linear fashion.

B. Effect of maxp

As we could choose *minth* to provide good performance in the previous subsection, we examined how *maxp* affected the queue length here although it has been fixed at 1/10 so far. Note that "*Out*" packets are dropped with probability 1.0 when the average queue length exceeds *maxth_out*. This leads to the throughput performance degradation as mentioned earlier. Therefore, *maxp* should be determined to avoid too long a queue. For this reason, *maxp* should not be too large. On the other hand, *maxp* should be also determined to prevent the buffer from becoming empty in order to use the bandwidth effectively. Thus, *maxp* should not be too small, and thus should be in a limited range.

Figure 4 illustrates the distribution of queue length in TCP Reno for several values of *maxp*, and Table 2 summarizes the related simulation results. From these results, we see that *maxp* greater than or equal to 1/10 can cause an empty buffer whereas 20 TCP flows share it at a time. This means that such a dropping probability is still rather high. On the other hand, the average queue length increases with *maxp*.

From Fig.4, the queue becomes empty less frequently with smaller *maxp* in TCP Reno. On the other hand, the queue becomes very long more frequently over time. Table 2 gives the corresponding numerical results: the probability of an empty buffer denoted by ql_zero, the average queue length denoted by ql_ave, the standard deviation denoted by ql_std and throughput.

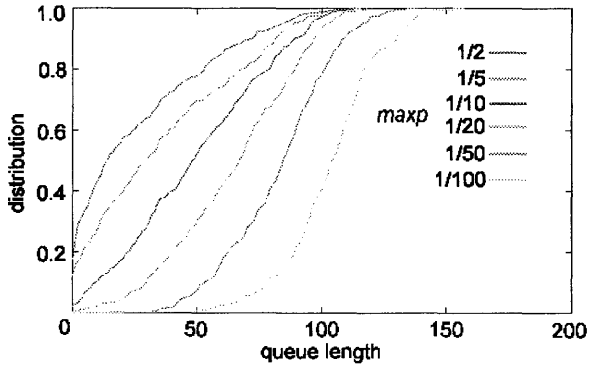

Fig.4 Queue length distribution in TCP Reno

From Table 2, *maxp* of 1/10 to 1/50 can attain shorter queue length as well as good throughput performance.

We have also executed similar simulations in TCP New Reno and TCP Sack, and the obtained results are shown in Table 2. The results are very similar to those of TCP Reno. In TCP New Reno and TCP Sack, the throughput performance is improved in case of *maxp* less than or equal to 1/10 compared with that of TCP Reno. Both TCP New Reno and TCP Sack are designed to use the network resource effectively, and thus allow longer queues in the buffer rather than TCP Reno. In the table, *maxp* of 1/10 to 1/50 can attain good throughput performance

Tbl.2 Queue length characteristics and throughput for different *maxp* in TCP Reno, TCP New Reno and TCP Sack

TCP	*maxp*	ql_ zero	ql_ ave	ql_ std	Throughput
Reno	1/2	0.170	26.08	28.14	63.18
	1/5	0.117	36.04	30.68	67.12
	1/10	0.020	49.72	28.69	68.89
	1/20	0.003	64.79	25.61	69.61
	1/50	0.000	83.24	20.99	69.76
	1/100	0.000	103.71	19.00	69.77
Newreno	1/2	0.117	35.64	29.73	66.02
	1/5	0.050	43.92	30.09	68.29
	1/10	0.030	53.20	28.12	69.20
	1/20	0.007	65.19	26.07	69.65
	1/50	0.000	86.16	20.00	69.76
	1/100	0.000	106.55	18.16	69.78
Sack	1/2	0.067	36.35	28.63	67.22
	1/5	0.040	44.54	29.58	68.46
	1/10	0.013	54.31	28.61	69.33
	1/20	0.003	66.99	23.05	69.73
	1/50	0.000	85.92	19.68	69.75
	1/100	0.000	107.79	19.88	69.77

C. *Effect of Delay in Dropping "Out" Packets*

So far, the dropping probability has depended on the average queue length, so called EWMA (Exponential Weighted Moving Average), as in RED. The average queue length (*avg*) is calculated in the following way.

$$avg = (1 - w_{qout})\, avg + w_{qout}\, qlen,$$

where *qlen* indicates the current queue length and w_{qout} is a weight factor used in calculating the average queue length. Paper [1] discusses the appropriate values for w_{qout}, and recommends 0.002 for it. EWMA functions as a low-pass filter, and prevents transient congestion from reacting on packet dropping. That works very well in RED, but there is another possible way in RIO since packets from a flow sending excess traffic can be differentiated by their marker of "*Out*."

Figure 5 shows the queue length at every time the packets arrive at the buffer and the average one derived by EWMA of w_{qout} = 0.002. The average queue length varies within a very limited range compared with the actual one, which packets arriving at the buffer really see. Some action based upon the average queue length is delayed so that it may not be often adequate in determining whether to drop arriving "*Out*" packets or not. That can allow a flow with excess traffic to send packets at a further higher rate although the queue length is already long or make some flows decrease their window size although the queue length is still very short.

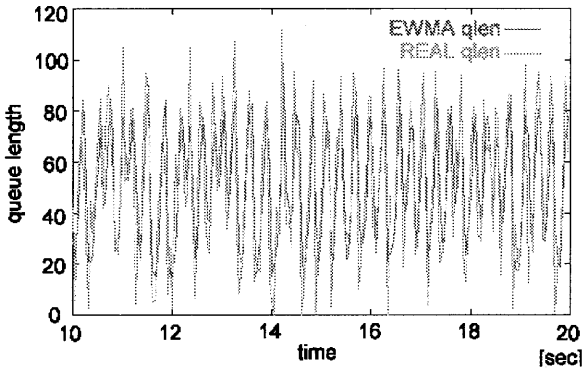

Fig.5 Queue length of EWMA and queue length (*maxp*=1/10)

Tbl. 3 Queue length characteristics for several w_{qout} In TCP Reno

maxp	w_{qout}	average	90 percentile	99 percentile
1/10	0.002	49	87	106
	0.010	53	78	94
	0.100	49	73	97
	1.000	49	71	95
1/20	0.002	65	94	110
	0.010	63	86	101
	0.100	59	80	96
	1.000	58	80	94
1/50	0.002	83	107	125
	0.010	82	102	117
	0.100	76	97	113
	1.000	76	97	114

Therefore, we examined the effect of w_{qout} on the performance while "*In*" packets were treated based upon the average queue length like in conventional RIO and

gave the simulation results in Table 3. w_{qout} = 0.002 is usually used in RED and RIO so far. When w_{qout} = 1, the probability of dropping "*Out*" packets depended on the current queue length without any delay. In the simulation, about 0.26 million packets were generated, and the obtained statistics were associated with the queue length, which those packets see in arriving at the buffer. The average, 90 percentile and 99 percentile queue length are given in Table 3. By managing the queue based upon the queue length with small delay, the average queue length become shorter and the queue length could be limited within a narrow range. The throughput performance is omitted in the table, but is not so different from each other. From these results, w_{qout} greater than 0.1 can be recommended.

Figure 6 shows the queue length dynamics in RIO based upon current queue length without delay, which is denoted by RIO with prompt feedback. The queue length is controlled very well within a narrow range in RIO with prompt feedback. We can see how RIO with prompt feedback can control the queue length very well by comparison of Fig.2 and Fig.6. Note that EWMA with w_{qout}=0.002 is used in Fig.2.

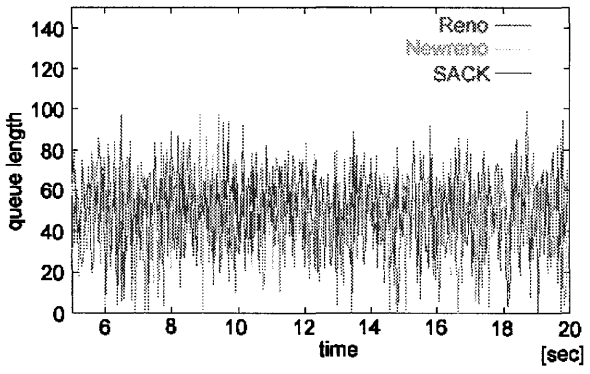

Fig.6 Queue length dynamics in RIO with prompt feedback

Furthermore, Table 4 gives the related simulation results; Table 4a and 4b at the cases with w_{qout} = 0.002 and w_{qout} = 1, respectively. By comparing these tables, we see that RIO with prompt feedback improves the throughput performance very well when *maxp* is 1/10 or more and decreases the average queue length as well as limiting the queue length within a narrow range when *maxp* is 1/20 or less.

By using the prompt feedback, the queue can be controlled in an appropriate way, which achieves high throughput even with small *maxth* such as 150. This further allows a buffer of small capacity. Moreover, the prompt feedback achieves a high throughput over a wide range of *maxth* and *maxp*, which leads to ease of parameterization.

Tbl.4a Various characteristics in case of w_{qout} = 0.002 in TCP Reno

maxth	*Maxp*	ql_ zero	ql_ ave	ql_ std	Through put
150	1/10	0.070	40.97	30.47	68.10
	1/20	0.017	54.22	27.81	69.40
	1/50	0.000	74.01	21.34	69.77
200	1/10	0.020	49.72	28.69	68.89
	1/20	0.003	64.79	25.61	69.61
	1/50	0.000	83.24	20.99	69.76
250	1/10	0.007	56.26	26.81	69.41
	1/20	0.000	71.25	22.55	69.73
	1/50	0.000	91.36	19.88	69.75

Tbl.4b Various characteristics under prompt feedback in TCP Reno

maxth	*Maxp*	ql_ zero	ql_ ave	ql_ std	Through put
150	1/10	0.023	44.52	21.33	69.38
	1/20	0.000	51.59	19.56	69.63
	1/50	0.000	67.27	18.13	69.75
200	1/10	0.007	49.89	19.28	69.60
	1/20	0.003	59.29	18.44	69.71
	1/50	0.000	77.42	18.33	69.77
250	1/10	0.000	53.75	19.46	69.66
	1/20	0.000	63.62	18.68	69.73
	1/50	0.000	87.36	18.71	69.76

4. Concluding Remarks

In this paper, we have studied the parameterization of RIO to achieve both high throughput and low delay performance in three TCP variants: TCP Reno, TCP New Reno and TCP Sack. We were able to

parameterize RIO for that purpose in terms of both *minth* and *maxp*. Furthermore, we have shown that dropping "Out" packets should depend upon the queue length without much delay. In fact, with the prompt feedback and *maxp*=1/10 or 1/20, we have obtained good throughput and a shorter queue length, which in turn required only a short buffer. From a different viewpoint, we were able to see that the prompt feedback enabled rather easy parameterization because it achieved good performance over a wide range of *maxth* and *maxp*.

We have performed other simulations and have obtained similar results in different contexts although we cannot put them here due to the lack of space. We will further study the performance of TCP sharing a link with EF (Expedited Forwarding) packets with priority in diffserv architecture, in which the available bandwidth is changing for TCP and thus the queue length can vary over a wide range. The prompt feedback will contribute to the improvement of RIO queue management scheme7. We have already examined that without AF PHB in [10].

5. Acknowledgement

We thank Tetsuya Takine, Kyoto University, for discussions on our simulation results.

This work was supported in part by Research for the Future Program of Japan Society for the Promotion of Science under the Project ``Integrated Network Architecture for Advanced Multimedia Application Systems'' (JSPS- RFTF97R16301), Telecommunication Advancement Organization of Japan under the Project ``Global Experimental Networks for Information Society Project,'' and a Grant-in Aid for Scientific Research (12450156) of the Ministry of Education, Science, Sports and Culture, Japan.

6. Bibliography

[1] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," ACM/IEEE Transactions on Networking, vol.1. no.4, pp. 397-413, August 1993

[2] K. Nichols, S.Blake, F. Baker, and D.L. Black, "Definition of the Differentiated Service Field (DS Field) in the IPv4 and IPv6 Headers," RFC2474, December 1998

[3] S. Black, D. Black, M. Carlson, E. Davies, Z. Wang and W. Weiss, "An Architecture for Differentiated Services," RFC2475, December 1998

[4] J. Heinanen, F. Baker, W. Weiss and J. Wroclawski, "Assured Forwarding PHB Group," RFC2597, June 1999

[5] V. Jacobson, K. Nichols and K. Poduri, "An Expedited Forwarding PHB," RFC2598, June 1999

[6] D. Clark and W. Fang, "Explicit Allocation of Best-Effort Packet Delivery Service," IEEE/ACM Transactions on Networking, Vol.6, No.4, pp.362-373, August 1998

[7] W. Fang and L. Peterson, "TCP mechanisms for Diff-Serv Architecture," Princeton University Technical Report 605-99, September 1999

[8] VINT Project, "Network Simulator version 2(ns-2)," http://www.isi.edu/nsnam/ns/

[9] Sean Murphy, "Diffserv additions to ns-2," http://www.teltec.dcu.ie/~murphys/ns-work/ diffserv/

[10] H. Koga, Y. Hori and Y. Oie, "Performance Comparison of TCP Implementations in QoS Provisioning Networks," 2D-3, INET2000, July 2000

[11] K. Fall and S. Floyd, "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP," Computer Communication Review, Vol. 26 No. 3, July 1996, pp. 5-21