Comparisons of Packet Scheduling Algorithms for Fair Service among Connections on the Internet

Go Hasegawa, Takahiro Matsuo, Masayuki Murata and Hideo Miyahara
‡Department of Infomatics and Mathematical Science
Graduate School of Engineering Science, Osaka University
1-3, Machikaneyama, Toyonaka, Osaka 560-8531, Japan
Phone: +81-6-850-6588, Fax: +81-6-850-6589
E-mail: {hasegawa,t-matuo,murata,miyahara}@ics.es.osaka-u.ac.jp

Abstract—**We investigate the performance of TCP under three representatives of packet scheduling algorithms at the router. Our main focus is to investigate how fair service can be provided for elastic applications sharing the link. Packet scheduling algorithms that we consider are FIFO (First In First Out), RED (Random Early Detection), and DRR (Deficit Round Robin). Through simulation and analysis results, we discuss the degree of achieved fairness in those scheduling algorithms. Furthermore, we propose a new algorithm which combines RED and DRR algorithms in order to prevent the unfairness property of the original DRR algorithm, which appears in some circumstances where we want to resolve the scalability problem of the DRR algorithm. In addition to TCP Reno version, we consider TCP Vegas to investigate its capability of providing the fairness. The results show that the principle of TCP Vegas conforms to DRR, but it cannot help improving the fairness among connections in FIFO and RED cases, which seems to be a substantial obstacle for the deployment of TCP Vegas.**

Keywords—**Fairness, FIFO (First In First Out), RED (Random Early Detection), DRR (Deficit Round Robin), TCP (Transmission Control Protocol)**

I. Introduction

The conventional Internet has only been providing the best effort service, and it could not offer throughput and/or delay guarantees. It is also lack of fairness guarantees; TCP connections sometimes receive unfair performance in terms of, e.g., throughput. See, e.g., [1]. However, we now need to provide commercial network services by the Internet. That is, a new service should be available within the network to support the *differentiated services* among the users [2]. Along with the context of *diff-serv* models, several service principles have recently been proposed; for example, a constant throughput may be preferred to some connections, or QoS support is necessary for real–time applications. For example, in [3], the authors have proposed an *Explicit Capacity* framework for allocating the network capacity to users in a controlled way even during congestion periods.

Another important service that the next–generation Internet should support is *fair allocation of the bandwidth*, which is our main subject of this paper. It is one of most desired features for elastic applications, but not supported by the current Internet, and we believe that it may be more important even than network efficiency. A one existing service found in the literature is the USD (User Share Differentiation) scheme described in [4], where users are provided different service qualities from ISPs (Internet Service Providers) based on the contracts. However, the authors in [4] do not provide a quantitative evaluation of USD to show how the users are *differentiated*. One promising way to realize the service differentiation for the elastic applications seems to be DRR (Deficit Round Robin) presented in [5] where the round robin scheduling is performed among active connections. In [5], an extensive evaluation of the DRR algorithm is provided, but they assume Poisson arrivals of packets from each connection. That is, the authors do not consider the behavior of the upper–layer protocol, i.e., TCP (Transmission Control Protocol).

In this paper, we focus on the degree of *fairness* provided to TCP connections by comparing three packet scheduling algorithms at the router. The first one is FIFO (First In First Out, or Drop–Tail), which is widely used in the current Internet routers because of its simplicity. The second is RED (Random Early Detection) [6], which drops incoming packets at a certain probability. While the original idea of the RED algorithm is to avoid consecutive dropping of packets belonging to the same connection, it also has a capability of achieving a fair service among connections by spreading packet losses. The last one is DRR, which is a more aggressive one in the sense that it actively maintains per–flow queueing for establishing fair service. For TCP, we consider the Reno version, which has widely been used in the current Internet. The Vegas version [7], adopting a different congestion control mechanism from TCP Reno for larger performance gain, is also considered.

In this paper, for reference purposes, we will first show simulation results that FIFO cannot provide fairness among connections at all because of a bursty nature of packet losses (see Subsection III-A). It is next shown that RED offers better fairness than FIFO to TCP Reno connections, but it cannot keep a good fairness when the capacity of shared link becomes small compared with the total input link capacity (Subsection III-B). In TCP Vegas, on the other hand, RED offers less fairness than FIFO because of the essential incompatibility of TCP Vegas to the RED algorithm (Section IV).

The packet scheduling algorithms and TCP versions that we will use in this paper are not new. Our main contributions in the current paper is that the properties mentioned above are also shown through analytical results. While the model used in the analysis is very simple, the basic features of the above scheduling algorithms can be well explained. From the analysis results, we further propose the enhanced version of RED algorithm, where we set each connection's packet dropping probability dependently on its input link capacity, to avoid the unfairness property of the original RED algorithm. Another enhancement method of RED can be found in [8], where the flow state are maintained for some degree of fairness enhancements.

The above method can be used to resolve an inherent problem of the DRR algorithm. DRR can provide almost perfect fairness among connections in both cases of TCP Reno (Subsection III-C) and Vegas (Subsection IV-C), but DRR requires per–flow queueing. Since we mainly consider the ISP model, we may not need to consider the *stateless* fair queueing mechanism such as the one found in [**?**]. However, DRR has a scalability problem in that as the number of subscribers grows, the larger number of queues becomes necessary. One possible solution is flow aggregation which treats several connections as a single flow of DRR. However, it results in that the fairness property of DRR becomes lost when multiple TCP connections are assigned to the same queue. Based on our analytical results, we last apply the RED mechanism to each queue of DRR (called DRR+) for fairness enhancement. We show that our DRR+ can provide a reasonably good fairness even compared with DRR through the simulation results (Subsection III-D).

For the discussions above, we use the network model where the uplink of the access line of ISP is shared by the subscribers with different capacities. The effect of the reverse traffic is also considered by the model where the downlink is shared by the subscribers. Although we will not show the results due to space limitation, we have found that our analysis results in this paper can be applied to the reverse traffic model without any modification. The similar model is treated in [9], but we consider RED and DRR as the packet scheduling algorithm in addition to the FIFO algorithm employed in [9]. Further, we devote the fairness aspects of packet scheduling algorithms which are not considered in [9].

This paper is organized as follows. In Section II, we describe the model treated in Section III and IV. The packet scheduling algorithms is first summarized in Subsection II-A. We will also explain the congestion control algorithm of TCP Reno and TCP Vegas by focusing on those congestion avoidance mechanisms in Subsection II-B. In Subsection II-C, we explain the network model we will use in analysis and simulation, and introduce the fairness measure considered in this paper in Subsection II-D. In Section III, we evaluate the packet scheduling algorithms described in Section II-A in the case of TCP Reno through the simulation and the analysis, and propose DRR+ for fairness improvement. We next consider the case of TCP Vegas in Section III. Finally, we present some concluding remarks and future works in Section V.

II. The Model

A. Packet Scheduling Algorithms

In what follows, we briefly summarize the three packet scheduling algorithms, FIFO, RED and DRR for the current paper to be self–contained.

A FIFO algorithm is widely used in the current Internet routers because of its simple implementation. The incoming packets are accepted in order of arrivals. When the buffer at the router becomes full, arriving packets are dropped. Therefore, packets belonging to a particular connection can sometimes suffer from bursty packet losses. Then, fast retransmit [10] implemented in TCP does not work effectively. It is also likely to introduce bursty transmission of packets [6], which often results in further packet losses.

The problem mentioned above is solved by RED [6]. The RED algorithm is designed to cooperate with congestion control mechanisms provided in TCP. In RED, the router observes the avarage queue size (buffer occupancy), and the packets arriving at the router are dropped with a certain probability.

The DRR algorithm [5] is an extension of the round robin algorithm to be suitable to treat the variable–sized packets. The buffer at the router is logically divided into multiple queues. The arriving packets of each connection are stored in the pre-assigned queue by using a hash function, and those are served in a round–robin fashion. A difference from the *pure* round robin algorithm is that the packets with variable length can be allowed to keep the fairness among connections. In DRR, the bandwidth not used in the round is preserved to be used in the next round if the packet is too large to be served in the current round.

B. Congestion Control Mechanisms of TCP

In this paper, we consider two versions of TCP; Reno and Vegas. TCP Reno is widely used in the current Internet. TCP Vegas is a recently proposed one in [7].

In TCP Reno, the window size *cwnd* (congestion window size) is cyclically changed. *cwnd* continues to be increased until segment loss occurs. TCP Reno has two phases in increasing *cwnd*; Slow Start Phase and Congestion Avoidance Phase. When an ACK segment is received by TCP at the server side at time $t + t_A$ [sec], $cwnd(t+t_A)$ is updated from $cwnd(t)$ as follows (see, e.g., [10]);

$$cwnd(t+t_A) = \begin{cases} cwnd(t)+1, & \text{if } cwnd(t) < ssth; \\ cwnd(t)+\dfrac{1}{cwnd(t)}, & \text{if } cwnd(t) \geq ssth; \end{cases} \tag{1}$$

where $ssth$ [segments] is the threshold value at which TCP changes its phase from Slow Start Phase to Congestion Avoidance Phase. When segment loss is detected by timeout or fast retransmission algorithm [10], $cwnd(t)$ and $ssth$ are updated as

$$ssth = cwnd(t)/2; cwnd(t) = ssth$$

In TCP Reno (and the older version Tahoe), the window size, *cwnd*, continues to be increased until segment loss occurs due to congestion. Then, the window size is throttled, which leads to the throughput degradation of the connection. However, it cannot be avoided because of an essential nature of the congestion control mechanism adopted in TCP Reno. That is, it can detect network congestion only by segment loss. However, throttling the window size is not adequate when the TCP connection itself causes the congestion because of its too large window size. If *cwnd* is appropriately controlled such that the segment loss does not occur in the network, the throughput degradation due to the throttled window can be avoided. This is the reason that TCP Vegas was introduced.

TCP Vegas employs another mechanism, in which it controls *cwnd* by observing changes of RTTs (Round Trip Time) of segments that the connection has sent before. If observed RTTs become large, TCP Vegas recognizes that the network begins to be congested, and throttles *cwnd* down. If RTTs become small, on the other hand, TCP Vegas determines that the network is

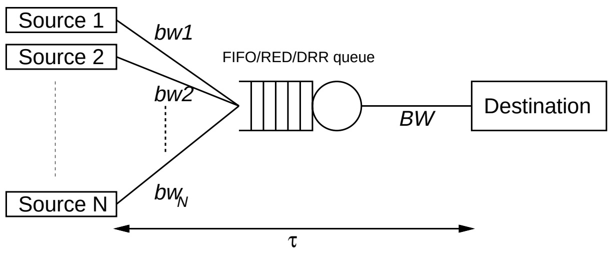


Fig. 1. Network model

relieved from the congestion, and increases *cwnd* again. Then, *cwnd* in an ideal situation becomes converged to the appropriate value. In Congestion Avoidance Phase, the window size is updated as;

$$cwnd(t+t_A) = \begin{cases} cwnd(t)+1, & \text{if } \mathit{diff} < \frac{\alpha}{base\_rtt} \\ cwnd(t), & \text{if } \frac{\alpha}{base\_rtt} \le \mathit{diff} \le \frac{\beta}{base\_rtt} \\ cwnd(t)-1, & \text{if } \frac{\beta}{base\_rtt} < \mathit{diff} \end{cases} \quad (2)$$
$$\mathit{diff} = cwnd(t)/base\_rtt - cwnd(t)/rtt$$

where rtt [sec] is an observed round trip time, $base\_rtt$ [sec] is the smallest value of observed RTTs, and α and β are some constant values. Note that Eq. (2) used in TCP Vegas indicates that if RTTs of the segments are stable, the window size remains unchanged.

C. Network Model

Recalling that our main purpose of the current paper is to investigate the fairness aspect of packet scheduling algorithms, we will use a simple network model as depicted in Figure 1.

There are the number N of connections between N sources (SES$_1$, SES$_2$, ..., SES$_N$) and one destination (DES). N connections share the bottleneck output link of the router. The capacity of the input link between the sources and the router are defined as bw_1, bw_2, ..., bw_N Kbps, and that of the output link between the router and destination is BW Kbps. We assume $bw_1 \le bw_2 \le \ldots \le bw_N$. By the above model, we intend to consider the uplink of the access line of the ISP, which is shared by the subscribers with different capacities.

In the following numerical examples throughout the paper, the propagation delay between SES_i and DES, τ, is identically set to be 100 msec. The buffer size of the router is 60 Kbytes. A TCP packet size is fixed at 2 Kbytes. Every sender is assumed to be a greedy source, that is, it has infinite packets to transmit. We also assume that in the case of DRR, the connection can be identified by the router so that the packets from the connection can be appropriately queued at the per–flow buffer at the router.

D. Definition of Fairness

We define the *fair* service by taking account of the input link capacity. Its simplest form is that the throughput is given in proportion to its input link capacity under the condition that the output link capacity is smaller than total of the input link capacities. That is, we say that a *good fairness is achieved* if the throughput of connection i, ρ_i, is given as

$$\rho_i = BW \cdot \frac{bw_i}{\sum_j bw_j}$$

We note that other definitions of the fairness can be considered. A more natural definition may be the function of subscription fees, which may be determined by (but not be proportional to) the input link capacity in the ISP model. We will not treat such a case for simplicity of presentation, but it is not difficult to incorporate it. For example, the weight factor is allowed to be arbitrary in the DRR case. The RED case can also be treated in this context by utilizing our analysis presented later.

III. The Case of TCP Reno

In this section, we consider TCP Reno to investigate the fairness property of three packet scheduling algorithms. In addition to the simulation results, we develop the analysis result for the RED scheduling algorithm. The analysis results supports observations on the fairness property of the RED algorithm obtained from the simulation results. We then investigate DRR to demonstrate its effectiveness through simulation experiments.

In what follows, we set four TCP connections which have different capacities of 64, 128, 256 and 512 Kbps. The output link capacity is varied from 400 Kbps to 960 Kbps to investigate the effect of the output link capacity on fairness. In the simulation results, we simulated 5,000 sec in each experiment to obtain the result, which approximately corresponds to 300,000 packet generation.

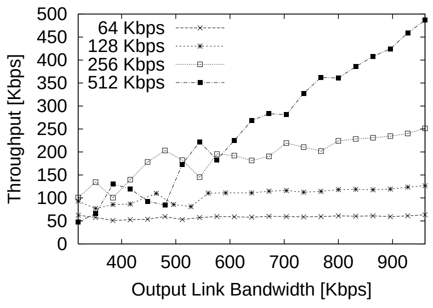
A. FIFO Case

We first show the FIFO case in terms of the average throughput during the simulation run (Figure 2(a)), the relative throughput (Figure 2(b)), and packet loss rate (Figure 2(c)) for all connections as a function of the output link capacity. Relative throughput means the ratio of the average throughput against the input link capacity. When all connections have identical relative throughput, it is said that the router perfectly provides fair service among connections in our definition. In Figure 2(a), the solid line labelled “total” shows the total throughput of four connections. From Figures 2(a) and 2(b), it is clear that fairness cannot be kept at all. In some region where the output link capacity is small, the throughput of the connection with smaller input link capacity is larger even than that of the connection with larger input link capacity. It can be explained as follows. In the FIFO algorithm, packet loss occurs independently of the packet arrival rate as shown in Figure 2(c), and the packet loss becomes bursty. Since the connection with larger input link capacity experiences a higher degree of burstiness of packet losses, its performance degradation becomes larger.

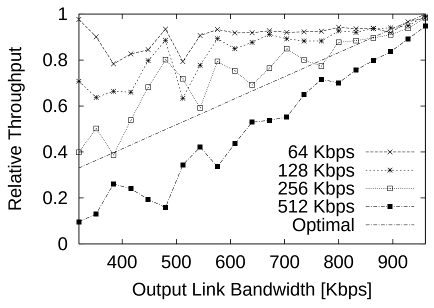
B. RED Case

B.1 Simulation Results

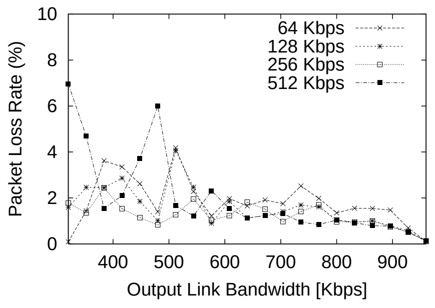
We next investigate the RED case. Recalling that the buffer size of the router is set to be 60 Kbyte, we set th_{min} = 10 Kbytes, th_{max} = 30 Kbytes and $p = 0.02$ in simulation. p shows the packet dropping probability defined in RED, with which incoming packets are dropped when the avarage queue length is over the threshold th_{min}. Figure 3 shows simulation results of the RED algorithm in that case. By comparing the “total” line in Figures 3(a) and 2(a), it can be observed that the RED algorithm can attain higher total throughput than that of the FIFO algorithm because RED can avoid bursty packet losses by dropping



(a) Average throughput

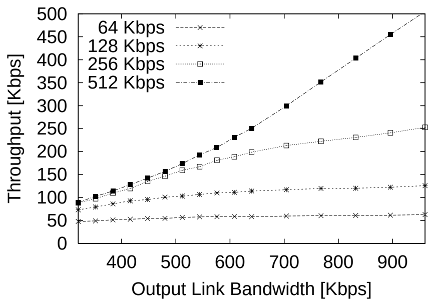


(b) Relative throughput

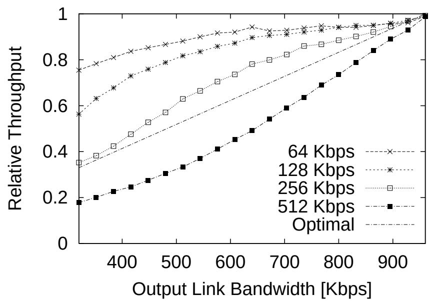


(c) Packet loss rate

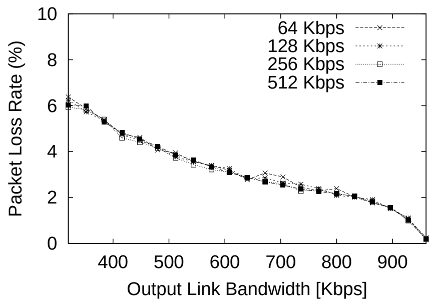
Fig. 2. FIFO case with TCP Reno



(a) Average throughput



(b) Relative throughput



(c) Packet loss rate

Fig. 3. RED case with TCP Reno

arriving packets with probability p, which results in that TCP's fast retransmit algorithm works effectively. However, if we focus on the fairness, it is clear that an improvement is very limited. It is especially true when the output link capacity is small; the throughput of all connections becomes almost identical (Figure 3(a)). Also, the packet loss rates of all connections are almost equal as shown in Figure 3(c). Of course, this is one of key features that the RED algorithm intends; the number of the lost packets of each connection can be kept in proportion to its input link capacity by its mechanism. The problem is that it leads to the unfairness treatment of connections with different capacities.

The above result is just one example. Also, it is questionable whether simulation time of 300,000 packets generation is adequate or not for examining the fairness degree. To examine its generality, we next show the analysis of the RED algorithm. Through analysis, it is proven that the unfairness observed in simulation is inherent in the RED algorithm.

B.2 Analysis Results and Discussions

We assume in the following analysis that there are N connections in the network (Figure 1) with the input link capacities of bw_1, bw_2, ..., bw_N [packets/sec], where $bw_1 \leq bw_2 \leq$, ..., $\leq bw_N$. We denote the packet dropping probability of the RED algorithm by p, and the propagation delay between sources and the destination by τ. We also assume that the average queue length is always larger than th_{min}, that is, all arriving packets are dropped with probability p. For analysis, we focus on TCP's typical *cycle* of the window size as shown in Figure 4; the cycle begins at the time when the previous packet loss occurs, and terminates when the next packet loss occurs. We consider that the cycle begins at time $t = 0$ [sec]. We do not take account of the slow start phase [10] since the objective of the RED algorithm is essentially

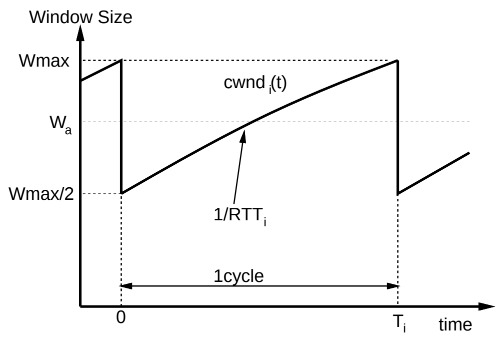


Fig. 4. TCP's cyclically change of the window size for connection i

to avoid falling into that phase.

Since all arriving packets are dropped at the router with probability p by our assumption, the connection can send $1/p$ packets in one cycle (between the events of packet losses). We define the number of packets transmitted during one cycle as N_p, that is, $N_p = 1/p$. During the cycle, the window size of connection i, $cwnd_i(t)$ [packets], is increased linearly since we only consider the congestion avoidance phase [10]. The window size is halved when packet loss detected by fast retransmit, and therefore $cwnd_i(t)$ is given as

$$cwnd_i(t) = \frac{W_{max}}{2} + \frac{1}{RTT_i} \cdot t, \qquad 1 \leq i \leq N, \tag{3}$$

where RTT_i [sec] is an average round trip time of packets for connection i, and W_{max} [packets] is the value of the window size at the time when packet loss occurs. Then, the following equation for the total number of the packets in one cycle should be

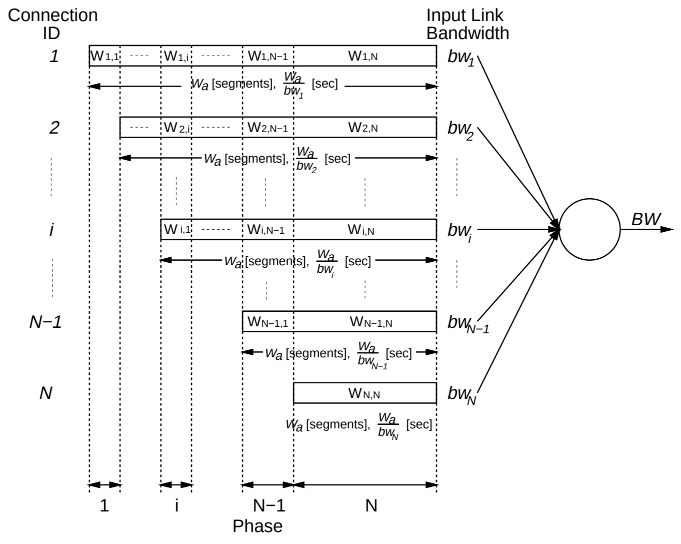

Connection ID
Input Link Bandwidth
1
W1,1
W1,i
W1,N−1
W1,N
bw1
Wa [segments], Wa/bw1 [sec]
2
W2,i
W2,N−1
W2,N
bw2
Wa [segments], Wa/bw2 [sec]
i
Wi,1
Wi,N−1
Wi,N
bwi
Wa [segments], Wa/bwi [sec]
N−1
WN−1,1
WN−1,N
bwN−1
Wa [segments], Wa/bwN−1 [sec]
N
WN,N
bwN
Wa [segments], Wa/bwN [sec]
BW
1
i
N−1
N
Phase

Fig. 5. Analysis of the RED algorithm

satisfied for connection i;

$$\int_0^{T_i} cwnd_i(t)dt = N_p, \qquad 1 \leq i \leq N, \tag{4}$$

where T_i is the time duration of the cycle as shown in Figure 4. From Eqs.(3) and (4), we can obtain W'_{max} [packets], the window size at the time when the next packet loss occurs, as

$$W'_{max} = \sqrt{{W_{max}}^2 + 2N_p} \tag{5}$$

From Eq.(5), we can obtain $\overline{W}_{max}$ [packets], the average value of W_{max} by equating W'_{max} and W_{max}. That is,

$$\overline{W}_{max} \simeq \sqrt{8/(3p)} \tag{6}$$

As a result, we derive W_a [packets], the average window size during the cycle as;

$$W_a = (3/4)\,\overline{W}_{max} \tag{7}$$

See Figure 4. From the equation above, we can see that the change of the window size does not depend on each connection's input link capacity, but on the packet dropping probability of the RED algorithm.

For further analysis, we make an assumption that each connection's window size is fixed at the average value, W_a. We then derive ρ_i, the throughput of connection i when W_a packets of its window are served at the router. To simplify the analysis, we consider the situation where all connections' first packets of the windows arrive at the router simultaneously as shown in Figure 5. In this figure, each square shows the *burst* of connection i's W_a segments, and its length represents the time duration $\frac{W_a}{bw_i}$ [sec]. Since all connections have different capacities bw_i on their links, it takes different time duration W_a/bw_i for all packets of connection i to arrive at the router as illustrated in Figure 5. That is, the segment burst of connection i is not served at the same rate, and it depends on the number of the connections sending simultaneously their packets. We divide all connections' packet burst into N 'phases' according to the number of connections which send the segments simultaneously. For example, since the number of connections transmitting their segments is i in phase i, the router processes segments of i connections at rate BW [segments/sec]. We denote the number of packets of connection i belonging to phase j by $W_{i,j}$ [packets] ($1 \leq i,j \leq N$). Since all segments in the phase are dealt at the rate in proportion to its input link bandwidth, we determine $W_{i,N}$ for phase N as follows;

$$\begin{aligned} W_{N,N} &= W_a \\ W_{i,N} &= W_{N,N} \cdot \frac{bw_i}{bw_N}, \qquad 1 \leq i \leq N. \end{aligned}$$

In the same manner, we can obtain all of $W_{i,j}$ by solving the following equations;

$$\begin{aligned} W_{j,j} &= W_a - \sum_{k=j+1}^{N} W_{j,k}, \qquad 1 \leq j \leq N-1, \\ W_{i,j} &= W_{j,j} \cdot \frac{bw_i}{bw_j}, \qquad 1 \leq j \leq N-1, 1 \leq i \leq j-1. \end{aligned}$$

The rate at which the packets are served at the router in phase j, S_j [packets/sec], must depend on the total capacity of the connections of phase j. Since, in phase j, all packets belonging to from connection 1 to connection j are served at the router, S_j is given as;

$$S_j = \begin{cases} BW, & \text{if} \sum_{k=1}^{j} bw_k > BW, \\ \sum_{k=1}^{j} bw_j, & \text{otherwise} \end{cases} \tag{8}$$

Therefore, the throughput of connection i during phase j, $R_{i,j}$, can be determined as follows;

$$R_{i,j} = W_{i,j} \Bigg/ \left(\frac{W_{i,j}}{\sum_{k=1}^{j} W_{k,j}} S_j \right) = \frac{\sum_{k=1}^{j} W_{k,j}}{S_j} \tag{9}$$

From Eqs.(8) and (9), ρ_i can be calculated as follows;

$$\rho_i = \sum_{k=N+1-i}^{N} \left(\frac{W_{i,j}}{W_a} R_{i,j} \right) \tag{10}$$

Although the RED algorithm can eliminate the bursty packet losses leading to TCP's retransmission timeout expiration, timeout expiration cannot be avoided perfectly [11]. Even if timeout expiration rarely happens, the effect of timeout expiration on throughput is large. Therefore, we next consider the throughput degradation caused by retransmission timeout expiration. We denote the probability of occurring timeout expiration in the window by P_{to}. We determine P_{to} according to the following simple equation;

$$P_{to} = \sum_{i=2}^{\infty} \binom{W_a}{i} \cdot p^i \cdot (1-p)^{W_a+1-i} \tag{11}$$

We assume that RTO_i [sec], the timeout duration for retransmission, becomes twice RTT_i, the Round Trip Time for connection i. RTT_i can be calculated by considering the effect of the

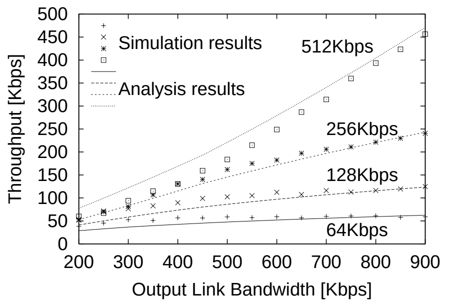


Fig. 6. Accuracies of analysis result in TCP Reno

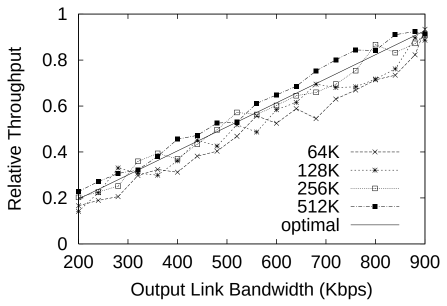


Fig. 7. The effect of enhanced RED

other connections' traffic;

$$RTT_i = 2\tau + \sum_{k \neq i} W_a/BW + W_a/\rho_i \qquad (12)$$

From these results, we finally have ρ'_i, the throughput of connection i, by considering the effect of TCP's retransmission timeouts;

$$\begin{aligned} \rho'_i &= (1 - P_{to}) \cdot \rho_i + P_{to} \cdot \frac{W_a/\rho_i}{W_a/\rho_i + RTO_i}\rho_i \\ &= \frac{\rho_i \cdot W_a + (1 - P_{to}) \cdot \rho_i^2 \cdot RTO_i}{W_a + \rho_i RTO_i} \qquad (13) \end{aligned}$$

Eq. (13) is obtained as follows. The first term $(1 - P_{to}) \cdot \rho_i$ represents the throughput without retransmission timeout, and the second term $\frac{W_a/\rho_i}{W_a/\rho_i + RTO_i}\rho_i$ is that with retransmission timeout. By Eq. (13), we can TCP throughput of each connection under RED algorithm, which takes account of the throughput degradation caused by TCP retransmission timeouts.

Figure 6 shows the throughput results from our analysis as a function of the output link capacity. In the figure, points represent the simulation results (which correspond to Figure 3(a)), and the lines show analysis results. We can observe from this figure that our analysis can give good agreements with simulation results, and that the unfairness property of the RED algorithm in the case of small output link capacity can be observed. This unfairness can be explained from the analysis result as follows. When the output link bandwidth becomes small, the rate at which the packets are served at the router of phase j becomes BW in almost all the phases. It is clearly shown in Eq. (8). That is, packets arriving at the router are served at rate BW, which results in that the throughput of all connections become equivalent. Furthermore, the connection whose input link bandwidth is larger can suffer from throughput degradation caused by TCP retransmission timeouts. This is also the reason why the throughput of the connection with the 512 [Kbyte/sec] input link bandwidth is largely degraded, which can be explained by Eq. (13).

B.3 Enhancement to RED

We last consider the enhancement to the RED algorithm (called *enhanced RED*) to avoid this unfairness by setting p dependently on each connection's input link capacity, according to the analysis results. We set p_i, which is the packet dropping probability of connection i, such that each connection's throughput becomes proportional to its input link capacity. The appropriate values of p_i's are calculated for all connections as follows.

1. Initialize p_i's.
2. Calculate ρ_i from the current p_i according to the analysis results. See Eq.(13).
3. If ρ_i is proportional to the input link capacity, set p_i to the current value.
4. If not, compare ρ_i with the ideal value, and adjust p_i of the connection having the largest difference between ρ_i and the ideal value. That is,
- If ρ_i is larger than the ideal value, change p_i to $a\,p_i$.
- If ρ_i is smaller than the ideal value, change p_i to $b\,p_i$.

The values of control parameters a and b that we will use in the following simulation are 1.1 and 0.9.

In the enhanced RED algorithm, we calculate p_i's for all connections according to the above algorithm. Figure 7 shows the simulation results on the relative throughput of the enhanced RED algorithm. Compared with Figure 3(b), it is clear that our enhanced version of the RED algorithm gives much better fairness than the original RED algorithm. In simulation, however, we set the control parameter values of a and b intuitively. It is a future research topic to seek an appropriate method to determine those parameters.

C. *DRR Case*

As explained in Subsection II-A, the router buffer is logically divided into several queues in DRR and each connection is assigned its own queue. We first consider the case where the large buffer is equipped with the router so that every connection is given a sufficient amount of buffer. In our model depicted Figure 8, four DRR queues are formed in the router, and DRR parameters are set such that each DRR queue is served in proportion to the input link capacity of the assigned connection.

Figure 10(a) shows the simulation results of relative throughput. Different from the FIFO (Figure 2) and RED (Figure 3) algorithms, the DRR algorithm provides very good fairness among connections even when the output link capacity is small. When the output link is large, on the other hand, the degree of the fairness is slightly degraded. It is because TCP's retransmission timeouts tends to frequently occur due to bursty packet loss at the queue since the FIFO discipline is used in each DRR queue. Then, the retransmission timeout degrades the performance more seriously. Thus the degree of performance degradation depends on the bandwidth–delay product of the connection. Furthermore, in the DRR algorithm, the capacity not used by a certain queue due to connection's retransmission timeout can be used by other connections. It increases the total throughput, but it is likely to lead to the unfairness among connections. This is

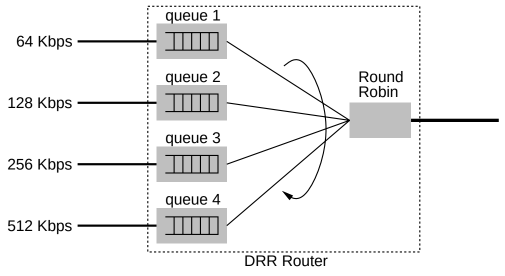


Fig. 8. Sufficient buffer case

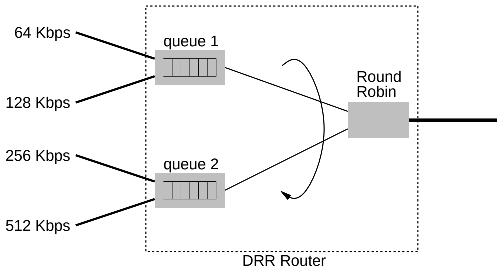
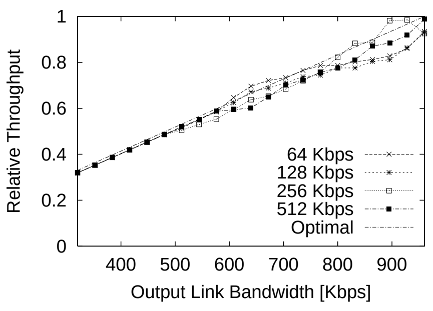
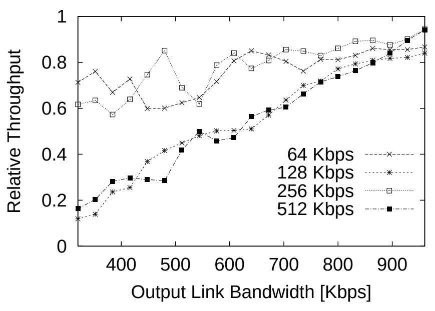


Fig. 9. Insufficient buffer case



(a) Sufficient buffer case



(b) Insufficient buffer case

Fig. 10. DRR case with TCP Reno

why fairness is degraded in the case of the large output link.

While the DRR algorithm assigns the DRR queues to each connection, several connections should be assigned to one DRR queue as the number of connections grows. It is because the number of DRR queues which can be prepared must be limited by the router buffer size and processing overhead. However, the performance of the DRR algorithm in such a case has not been known. For investigating such an insufficient buffer case, we assume that there are two queues and four connections, and each connection is assigned to the queue as shown in Figure 9. The 64 Kbps and 128 Kbps connections are assigned to one queue (queue 1 in the figure) and the 256 Kbps and 512 Kbps connections to another queue (queue 2). Each queue is assumed to be served in proportion to the total capacity of the assigned connections.

We show the simulation results in the insufficient buffer case in Figure 10(b) for the relative throughput. The buffer sizes of two queues are equivalently set to be 30 Kbytes. The lines labeled 'total-1' and 'total-2' indicate total throughput of two queues, queue 1 and queue 2. Although each queue is served in proportion to the total capacity of the assigned connections, the two connections assigned to the same queue show unfair throughput. This is because we assumed that the arriving packets are served according to a simple FIFO discipline within the DRR queue. As described in Subsection III-A, the FIFO algorithm cannot keep fairness among connection at all.

In this subsection, we have observed that the DRR algorithm gives much better fairness than FIFO and RED algorithms, but its fairness property is sometimes lost as each connection has different capacity or when multiple connections are assigned to one DRR queue. We henceforth consider to improve the fairness property of the DRR algorithm in the next subsection.

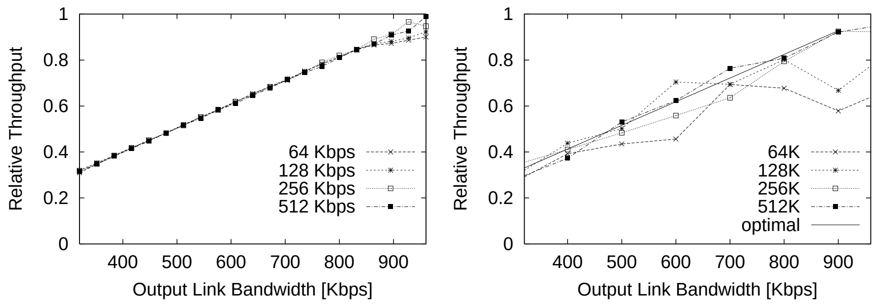
D. DRR+ Case

In the previous subsection, we have shown that the DRR algorithm has some unfairness property. The main reason was that each DRR queue serves packets by the FIFO discipline. In this subsection, we show some simulation results of DRR+, where the RED algorithm is applied to each DRR queue to prevent unfairness. In simulation, we consider both sufficient/insufficient buffer case. Note that, in the insufficient buffer case, we apply the enhanced RED algorithm to two DRR queues depicted in Figure 9. That is, in each queue, we set the assigned connections' packet dropping probabilities according to the enhanced RED algorithm in Subsection III-B.

Figure 11 shows the simulation results on the relative throughput. Our proposed method keeps good fairness in the sufficient buffer case (Figure 11(a)). Furthermore, when Figure 11(b) is compared with Figure 10(b), the fairness is significantly improved even in the insufficient buffer case.

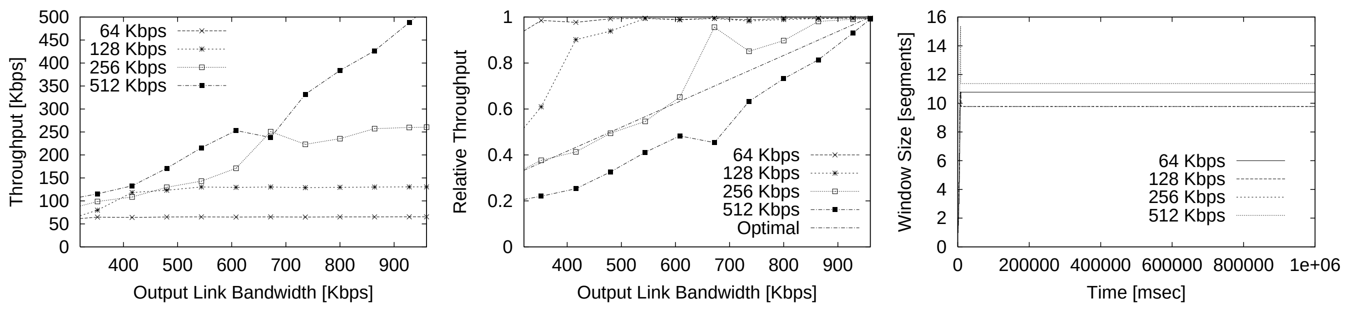
IV. TCP VEGAS CASE

In this section, we change the version of TCP to TCP Vegas to investigate the fairness property of three packet scheduling algorithms. TCP vegas conjectures the available bandwidth for the connection, and therefore its principle is likely to be well fit to the DRR algorithm. On the other hand, the RED algorithm does not help improve the fairness when TCP Vegas is employed since each connection's window size is not dominated by the packet dropping probability of the RED algorithm, but by the essential algorithm of TCP Vegas. The purpose of this section is to confirm the above observations.



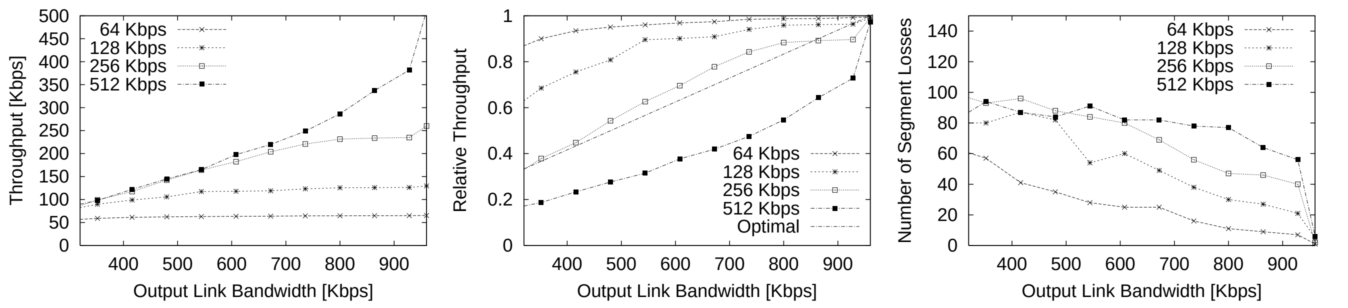
(a) Sufficient buffer case (b) Insufficient buffer case

Fig. 11. DRR+ case with TCP Reno



(a) Throughput (b) Relative throughput (c) Change of window sizes

Fig. 12. FIFO case with TCP Vegas



(a) Throughput (b) Relative throughput (c) Number of segment losses

Fig. 13. RED case with TCP Vegas

A. FIFO Case

Figure 12 plots simulation results of the FIFO case using TCP Vegas. Note that we omit the graph showing the number of packet loss since no segment loss was observed at the FIFO buffer. Compared with the TCP Reno case (Figure 2), it is clear that TCP vegas provides less fairness than TCP Reno. Especially, the connection with has smaller input link bandwidth achieve almost 100% throughput (Figure 12(b)). This unfairness property is caused by the essential characteristic of TCP Vegas. In TCP Vegas, no segment loss occurs at the router buffer if the network is stable, because the window size of all connection converges to certain values (Figure 12(c)). In Figure 12(c), it is noticeable that the converged window size is independent on each connection's input link bandwidth because $base\_rtt$ of each connection is almost equal (See Subsection II-B). In the current simulation setting, the converged window size is enough large for connections having smaller input link bandwidth to utilize its *bandwidth–delay-product*, but it is too small for connections with larger input link bandwidth. Therefore, while the result depends on the network environment, TCP Vegas sometimes fails to achieve fairness among connections due to the essential nature of its congestion control mechanism.

B. The RED Case

We next show the simulation results of the RED case in Figure 12. As in the case of TCP Reno (Subsection III-B), the fairness is slightly improved when compared with the FIFO case (Figure 12(b)). However, there still be significant unfairness among connections. This can be explained by the throughput analysis presented in the below. In the following analysis, we use the same notations as those introduced in Subsection III-B.

At the moment, we consider the situation where no segment loss occurs at the router, and each connection's windowsize converges to a certain value. The packet dropping of the RED will be considered later.

Let l_i [segments] be the number of connection i's segments in the router buffer, and $L = l_1 + \cdots + l_N$. Assume that each connection's throughput ρ_i [segments/sec] is proportional to the avarage number of its segments in the router buffer. This assumption is reasonable when the FIFO discipline is applied at the router buffer. Then, the following equation with respect to ρ_i is satisfied;

$$\rho_i = \min\left(bw_i, (l_i/L)BW\right) \tag{14}$$

According to the algorithm of TCP Vegas (Eq. (2)), we obtain;

$$\frac{\alpha}{base\_rtt_i} < \frac{W_i}{base\_rtt_i} - \frac{W_i}{rtt_i} < \frac{\beta}{base\_rtt_i} \tag{15}$$

$$\begin{aligned} base\_rtt_i &= 2\tau + 1/BW && (16)\\ rtt_i &= 2\tau + l_i/\rho_i && (17)\\ W_i &= 2\tau\rho_i + l_i = rtt_i \cdot \rho_i && (18) \end{aligned}$$

where rtt_i [sec] and W_i [segments] are the RTT and the window size of the connection i, respectively. $base\_rtt_i$ [sec] corresponds to $base\_rtt$ of connection i, which is the minimum value of RTTs of the connection. By substituting Eqs. (16)–(18) into Eq. (15), we obtain the following equation;

$$\alpha + \rho_i/BW < l_i < \beta + \rho_i/BW \tag{19}$$

From Eq. (19), $L\ (= l_1 + \cdots + l_N)$ can be calculated as follows;

$$\begin{aligned} &N\alpha + \sum_{j=1}^{N} \frac{\rho_i}{BW} < l_1 + \cdots + l_N < N\beta + \sum_{j=1}^{N} \frac{\rho_i}{BW} \\ &N\alpha + \sum_{j=1}^{N} \frac{\rho_i}{BW} < L < N\beta + \sum_{j=1}^{N} \frac{\rho_i}{BW} && (20) \end{aligned}$$

Recalling that $bw_1 \le bw_2 \le \ldots \le bw_N$, Eq. (14) yields

$$\rho_i = \begin{cases} bw_i & 1 \le i \le M \\ (l_i/L)BW & M+1 \le i \le N \end{cases} \tag{21}$$

Then, from Eqs. (19)–(21), we obtain ρ_i for $M+1 \le i \le N$ as follows;

$$\rho_i = \frac{l_i}{L - \sum_{j=1}^{M} l_i}\left(BW - \sum_{j=1}^{M} \rho_i\right), M+1 \le i \le N \tag{22}$$

Therefore, W_i, which is the converged window size of connection i, can be obtained by substituting Eq. (19) and Eq. (22) to Eq. (18).

In the above derivation, however, we do not take account of random segment losses adopted in the RED algorithm. We next consider the effect of throughput degradation caused by probabilistic segment loss of the RED algorithm. Although each connection's window size is controlled to be converged to a certain value in TCP Vegas, it is sometimes decreased by segment loss by the RED algorithm. We assume that the segment loss can be detected by the fast retransmit algorithm. Then, if the segment loss occurs after the window size reaches W_i, the window size is halved to $W_i/2$. That is, if $W_i/2 < 2\tau\rho_i$, the throughput is degraded until the window size reaches $2\tau\rho_i$. In Figure 14, we define 'one cycle' to be the time duration between two segment losses caused by RED. One cycle is divided into three phases; phase 1, phase 2, and phase 3 as in Figure 14. In phase 1, the window size is increasing according to the TCP Vegas's algorithm, but the window size is less than $2\tau\rho_i$. That is, the throughput is degraded by the segment loss during phase 1. In phase 2, the window size continues to increase as in phase 1, but the window size is larger than $2\tau\rho_i$ and there is no throughput degradation. In phase 3, the window size reaches the converged value, which is obtained from Eq. (18). It remains unchanged until the packet loss occurs at the end of this phase.

Let T_i [sec] and A_i [segments] be the time duration of phase i, and the number of transmitted segments in phase i, respectively. Furthermore, we introduce $\overline{\rho}_{i,j}$ [segments/sec] as the avarage throughput of connection i during phase j.

In phase 1 and phase 2, the ratio of window size increasing is $1/rtt_i$ [segments/sec] because the window size is increased according to TCP Vegas's congestion avoidance algorithm formulated by Eq. (2). Therefore, $\overline{\rho}_{i,1}$ is;

$$\overline{\rho}_{i,1} = \left(\frac{\frac{W_i}{2} + 2\tau\rho i}{2}\right) \Big/ \left(2\tau + \frac{1}{\rho_i}\right) \tag{23}$$

Because there is no throughput degradation in phase 2 and phase 3, $\overline{\rho}_{i,2}$ and $\overline{\rho}_{i,3}$ are identical to ρ_i, i.e.,

$$\overline{\rho}_{i,2} = \overline{\rho}_{i,3} = \rho_i \tag{24}$$

Since the increased rate of window size is $1/rtt_i$ [segments/sec], T_1 and T_2 can be calculated as follows;

$$\begin{aligned} T_1 &= \left(2\tau\rho_i - \frac{W_i}{2}\right) \cdot rtt_i && (25)\\ T_2 &= (W_i - 2\tau\rho_i) \cdot rtt_i && (26) \end{aligned}$$

A_1 and A_2 can also be calculated as follows;

$$\begin{aligned} A_1 &= \frac{1}{2}\left(2\tau\rho_i + \frac{W_i}{2}\right)\left(2\tau\rho_i - \frac{W_i}{2}\right) && (27)\\ A_2 &= \frac{1}{2}(W_i + 2\tau\rho_i)(W_i - 2\tau\rho_i) && (28) \end{aligned}$$

In phase 3, the window size is converged to W_i, and segment loss occurs at the router caused by the RED algorithm at the end of this phase. Since the avarage number of transmitted segments during 1 cycle is $(1/p)$, A_3 and T_3 can be obtained as;

$$\begin{aligned} A_3 &= 1/p - A_1 - A_2 && (29)\\ T_3 &= (A_3/W_i) \cdot rtt_i && (30) \end{aligned}$$

Finally, we can obtain $\hat{\rho}_i$, the throughput of connection i from Eqs. (23)– (26), (30) as follows;

$$\hat{\rho}_i = \frac{T_1\overline{\rho}_{i,1} + T_2\overline{\rho}_{i,2} + T_3\overline{\rho}_{i,3}}{T_1 + T_2 + T_3} \tag{31}$$

Figure 15 shows the result of the analysis as a function of the output link capacity. Compare with Figure 6. Our analysis again gives good agreements with simulation results, and it confirms the unfairness property of TCP Vegas when applied to the

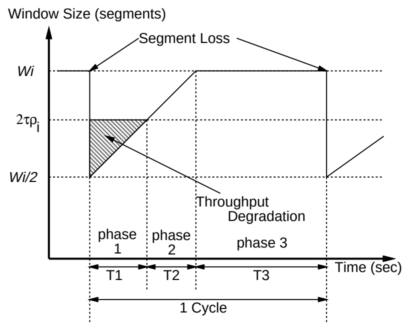


Fig. 14. Throughput degradation with RED segment loss

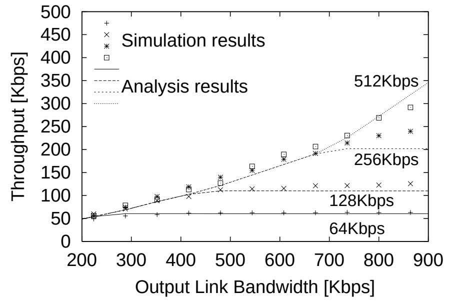


Fig. 15. Accuracies of analysis result in TCP Vegas

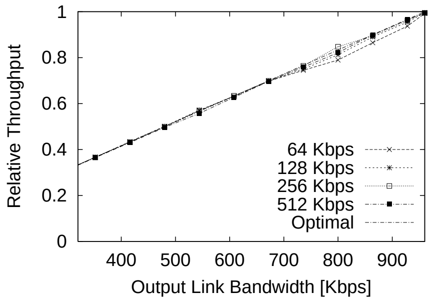


Fig. 16. DRR case with TCP Vegas

RED algorithm. In TCP Reno (Subsection III-B), we could improve the fairness by setting p (the packet dropping probability) dependently on each connection's input link capacity according to the analysis results. In TCP Vegas, however, we cannot apply it because the converged window size is independent on p as shown in Eqs. (18). That is, we cannot control each connection's throughput by p. Therefore, if we want to remove the unfairness property in the RED algorithm with TCP Vegas, we may have to give some modifications to the algorithm of TCP Vegas itself. Otherwise, we need to use the DRR algorithm as will be presented in the next subsection.

C. The DRR Case

Figure 16 shows the case of DRR. It can be observed from the figure that fairness among connections is fairly good (Figure 16), and better than TCP Reno case (Figure 10(a)). With TCP Reno, some connections could not utilize all amount of bandwidth assigned by the DRR mechanism due to segment loss. With TCP Vegas, on the other hand, no segment loss occurs at the router buffer, and then each connection can completely utilize the bandwidth assigned by the DRR mechanism. However, as the number of connections becomes large, the scalability problem is introduced as having been explained in Subsection III-C. In Subsection III-D, we have succeeded to avoid the unfairness by applying the RED mechanism to each DRR queue. In the current case, however, we cannot apply it because of the essential incompatibility of TCP Vegas to the RED algorithm as explained in Subsection IV-B. We need further investigation on this problem.

V. Concluding Remarks

In this paper, we have evaluated the performance of the router packet scheduling algorithms for fair service among connections through the simulation and analysis. We have obtained the following results on TCP Reno version; the FIFO algorithm cannot keep fairness among connections at all. The RED algorithm can improve fairness to some degree, but it fails to keep fairness in the different capacity case. The DRR algorithm offers better fairness than the FIFO algorithm and the RED algorithm, but its fairness property is lost when each connection has different capacity and/or when multiple connections are assigned to one DRR queue. Accordingly, we have proposed the DRR+ algorithm, where the RED algorithm is applied to each DRR queue to prevent unfairness, and show that it can improve fairness among connections in the different capacity case. We have also investigated the effect of TCP Vegas, which is expected to get higher throughput than TCP Reno, and have made clear through the simulation and analysis results that TCP Vegas cannot help improving the fairness among connections in FIFO and RED cases.

TCP Vegas has a good feature to attain the better performance than TCP Reno, as discussed in Section IV. However, it fails to keep the good fairness among connections with different input (and output) line capacities. For TCP Vegas to be introduced in the future Internet where the RED algorithm is widely deployed, the algorithm of TCP Vegas should be modified to improve the fairness among connections, which is a future research topic.

Acknowledgements

This work was partly supported 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".

References

[1] Go Hasegawa, Masayuki Murata, and Hideo Miyahara, "Fairness and stability of the congestion control mechanism of TCP," *Proceedings of IEEE INFOCOM'99*, March 1999.
[2] Diffserv Home Page,, ," `http://diffserv.lcs.mit.edu/`.
[3] David D. Clark and Wenjia Fang, "Explicit allocation of best effort packet delivery service," *available at* `http://diffserv.lcs.mit.edu/Papers/exp-alloc-ddc-wf.ps`, 1998.
[4] Zheng Wang, "Toward scalable bandwidth allocation on the internet," *On The Internet*, pp. 24–32, May/June 1998.
[5] M. Shreedhar and George Varghese, "Efficient fair queuing using deficit round robin," *IEEE/ACM Transactions on Networking*, vol. 4, no. 3, pp. 375–385, June 1996.
[6] Sally Floyd and Van Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, August 1993.
[7] Lawrence S. Brakmo and Larry L. Peterson, "TCP Vegas: End to end congestion avoidance on a global internet," *IEEE Jounal on Selected Areas in Communications*, vol. 13, no. 8, pp. 1465–1480, October 1995.
[8] D. Lin and R. Morris, "Dynamics of random early detection," *Proceedings of SIGCOMM'97*, pp. 127–137, October 1997.
[9] L. Neidhardt D. P. Heyman, T. V. Lakshman, "A new method for analyzing feedback–based protocols with applications to engineering web traffic over the internet," *Proceedings of IEEE SIGMETRICS'97*, pp. 24–38, February 1997.
[10] W. Richard Stevens, *TCP/IP Illustrated, Volume 1: The Protocols*, Addison-Wesley, Reading, Massachusetts, 1994.
[11] K. Fall and S. Floyd, "Simulation–based comparisons of Tahoe, Reno, and SACK TCP," *ACM SIGCOMM Computer Communication Review*, vol. 26, no. 2, pp. 5–21, July 1996.