PAPER Special Section on Architectures, Protocols, and Applications for the Future Internet

A New TCP Congestion Control Supporting RTT-Fairness

Kazumine OGURA[†a)], **Yohei NEMOTO**[†], ***Student Members***, **Zhou SU**[†], ***and*** **Jiro KATTO**[†], ***Members***

SUMMARY This paper focuses on RTT-fairness of multiple TCP flows over the Internet, and proposes a new TCP congestion control named "HRF (Hybrid RTT-Fair)-TCP". Today, it is a serious problem that the flows having smaller RTT utilize more bandwidth than others when multiple flows having different RTT values compete in the same network. This means that a user with longer RTT may not be able to obtain sufficient bandwidth by the current methods. This RTT fairness issue has been discussed in many TCP papers. An example is CR (Constant Rate) algorithm, which achieves RTT-fairness by multiplying the square of RTT value in its window increment phase against TCP-Reno. However, the method halves its windows size same as TCP-Reno when a packet loss is detected. This makes worse its efficiency in certain network cases. On the other hand, recent proposed TCP versions essentially require throughput efficiency and TCP-friendliness with TCP-Reno. Therefore, we try to keep these advantages in our TCP design in addition to RTT-fairness. In this paper, we make intuitive analytical models in which we separate resource utilization processes into two cases: utilization of bottleneck link capacity and that of buffer space at the bottleneck link router. These models take into account three characteristic algorithms (Reno, Constant Rate, Constant Increase) in window increment phase where a sender receives an acknowledgement successfully. Their validity is proved by both simulations and implementations. From these analyses, we propose HRF-TCP which switches two modes according to observed RTT values and achieves RTT fairness. Experiments are carried out to validate the proposed method. Finally, HRF-TCP outperforms conventional methods in RTT-fairness, efficiency and friendliness with TCP-Reno.

key words: *transport protocol, RTT-fairness*

1. Introduction

TCP (Transmission Control Protocol) is widely used in the current network and provides end-to-end, reliable data transport. The majority of data services from web surfing to HTTP multimedia streaming (like YouTube and P2P streaming) in the Internet are carried by TCP. In principle, an AIMD (Additive Increase and Multiplicative Decrease) mechanism [1] had been widely adopted. However, since the AIMD mechanism of original TCP-Reno autonomously determines a sending rate according to the self-clocking principle, it is well-known that it suffers from RTT (Round Trip Time) unfairness [2]–[7]. Therefore, RTT-fairness had been focused by many TCP papers such as TCP-Libra [3], BIC-TCP [6], CUBIC-TCP [7] and FAST-TCP [8].

CUBIC-TCP, which is an advanced version of BIC-TCP, is designed to support RTT-fairness, efficiency and TCP-friendliness. This protocol has a unique window control using elapsed time from a last packet loss into its window increment phase. This control indeed alleviates RTT-unfairness, but is not sufficient as shown in our past work [9], [15]. FAST-TCP improves RTT-fairness by its window size updating phase using RTT value as a parameter. However, it has a critical shortcoming when competing with TCP-Reno and suffers from throughput degradation [15].

In the classical work [4], CR (Constant Rate) was proposed to achieve RTT-fairness by incorporating the square of RTT value into its window increase algorithm. Its characteristics were studied in [5], and TCP-Libra was proposed as its extension. TCP-Libra adds components that lower the throughput variance and adapt its scalability in window increment phase according to link capacity. However, same as CR, TCP-Libra does not change its window decrement phase practically, namely using halved window size. This leads to inefficient network utilization over lossy, long fat (high-speed and long delay) or small buffer size network. Therefore, our TCP applies a clever algorithm (TCP-Westwood [10]) as its window decrement phase.

On the other hand, recent implemented TCP versions, as represented by CUBIC-TCP (for Linux OS) and Compound-TCP (for Windows OS) [11], are designed for throughput efficiency and TCP-friendliness with TCP-Reno. Various hybrid (loss/delay-based) TCPs [11]–[15] have been proposed for this purpose. Our TCP also belongs to hybrid TCPs and tries to keep their advantages.

In this paper, we make analytical models for achievement of RTT-fairness using the algorithm. In the models, we separate resource utilization processes into two cases. One is to utilize the bottleneck link capacity, and the other is to utilize the buffer space of the bottleneck router. From the model, we then propose a new congestion control named HRF (Hybrid RTT-Fair)-TCP to share these resources fairly in both cases. We combine the ideas of CR, CI (Constant Increase, refer [4]) and TCP-Westwood (TCPW) [10] into its method. HRF switches two modes (CR and CI) according to observed RTT values. Through simulation and implementation results, we will show HRF brings much better performances in RTT-fairness, throughput efficiency and friendliness with TCP-Reno.

This paper is organized as follows: Section 2 presents research background. Section 3 explains our analysis model on RTT-fairness according to buffer utilization status. Section 4 introduces our proposal, and Sect. 5 demonstrates experimental results. Finally, Sect. 6 provides conclusions of this paper.

Manuscript received May 16, 2011.
Manuscript revised September 20, 2011.

[†]The authors are with Waseda University, Tokyo, 169–0072 Japan.
a) E-mail: ogura@katto.comm.waseda.ac.jp

DOI: 10.1587/transinf.E95.D.523

2. Research Backgrounds

In this section, we explain RTT-unfairness of the AIMD congestion control, TCP-Westwood, CR and CI, respectively. In the following sections, we define "window increment phase" as window size updating phase with successful acknowledgement, and "window decrement phase" as window size updating phase with 3 duplicate acknowledgements (a packet loss happens).

2.1 RTT-Unfairness of AIMD Mechanism

A window increase rate of the AIMD congestion control based on TCP-Reno is inverse proportional to RTT values in principle. For example, [6] provides an analytical result of RTT unfairness of the AIMD congestion control, in which throughput ratio of two TCP flows having different RTT values is given by

$$\frac{throughput_1}{throughput_2} \propto \left(\frac{RTT_2}{RTT_1}\right)^{\frac{1}{1-d}} \tag{1}$$

where $throughput_i$ is an average throughput of flow i ($i = 1, 2$), RTT_i is an average RTT of flow i, and d is a constant which is determined by the congestion control mechanisms (e.g. d is 0.5 for TCP-Reno, 0.5∼1.0 for BIC-TCP, 0.82 for High-speed TCP and 1.0 for Scalable TCP). This equation proves RTT unfairness, according to which TCP flows with smaller RTT values expel TCP flows with longer RTT values.

2.2 TCP-Westwood

TCPW-RE (Rate Estimation) [10] improves throughput efficiency in window decrement phase. In window increment phase, the behavior of TCPW is the same as TCP-Reno. Window decrement phase of this protocol is expressed by

$$cwnd = max\left(\frac{RTT_{min}}{RTT} \cdot cwnd, \frac{cwnd}{2}\right) \tag{2}$$

where RTT_{min} and RTT are the minimum RTT and RTT just before the packet loss, respectively. The first term of Eq. (2) just clears a buffer of a bottleneck router instead of halving congestion window and causes no vacant capacity. If $cwnd * RTT_{min}/RTT$ is smaller than $cwnd/2$, TCP-W selects cwnd/2 not to decrease congestion window size less than TCP-Reno. Due to this fact, TCPW-RE can achieve better throughput efficiency than TCP-Reno.

2.3 CR (Constant Rate)

CR (Constant Rate), which was briefly introduced in [4], achieves RTT-fairness by multiplying the square of RTT to a window increment value. Its window increment/decrement phase are shown by

$$cwnd+ = k \cdot RTT^2 \quad [per\ RTT] \tag{3}$$

$$cwnd = cwnd/2 \tag{4}$$

where k is a constant and decide aggressiveness of this protocol. TCP flows using CR algorithm increase their window sizes by $k * RTT$ packets (or bytes) in one second and their throughput by k packets/sec (or bytes/sec) irrespective of RTT. Eq. (4) is the same window decrease strategy as Reno.

2.4 CI (Constant Increase)

CI (Constant Increase), which was also briefly mentioned in [4], is a linear version of CR's window update algorithm. In this paper, we define CI's window increase behavior in congestion avoidance phase by

$$cwnd+ = k' \cdot RTT \quad [per\ RTT] \tag{5}$$

where k' is similar to k in Eq. (3). TCP flows using CI algorithm increase their window sizes by k' packets (or bytes) in one second irrespective of RTT and their throughput by k'/RTT packets/sec (or bytes/sec). This protocol halves its window size same as Eq. (4) when a packet loss is detected.

3. Model Definition

We analyze TCP flows' behaviors until a packet loss happens by overflow. Assume the network has a single bottleneck link. Let B [pkt/s], R [pkt/s], RTT_{min} [s], S [pkt] and n represent the bottleneck link capacity, residual link capacity, minimum RTT value without buffering delay, the buffer size of a bottleneck router and the number of competing flows, respectively. In this model, we consider three characteristic protocols (TCP-Reno, CR and CI), but omit TCP-Westwood because TCP-Westwood behaves in the same way as TCP-Reno in window increment phase. x_{RTT} stands for the increment amount of the congestion window size per RTT. Table 1 shows x_{RTT} [pkt/RTT] of each protocol.

We separate the model into two cases according to the bottleneck router buffer status. Figure 1 shows the window size behavior of TCP-Reno. Model I (before buffering) is the case where there is residual capacity and no buffering delay. In this case, window size is always less than BDP (Bandwidth Delay Product), which is calculated by $RTT_{min} * B$. Model II (after buffering) is the case in which packet buffering is carried out at the bottleneck link router and a packet loss happens due to overflow. We evaluate window size behaviors of these two cases in the following subsections.

3.1 Model I: Vacant Case

In Model I, there is residual capacity on the bottleneck link

Table 1 Increment amount of window size of each protocol.

	TCP-Reno	CR	CI
x_{RTT}	1	$k*RTT^2$	$k'*RTT$

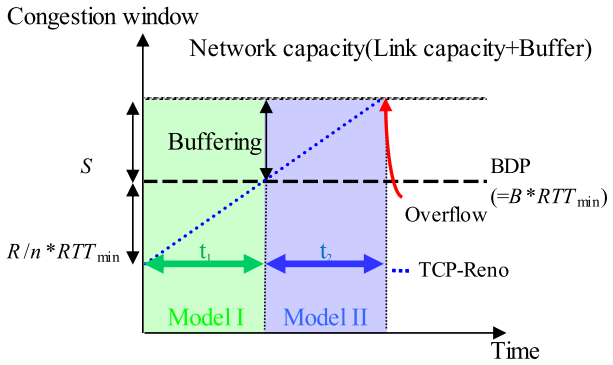


Fig. 1 The window size behavior of TCP-Reno.

Table 2 Time to fill residual capacity of each protocol.

	TCP-Reno	CR	CI
t_1	$\frac{R \cdot RTT_{min}^2}{n}$	$\frac{R}{n \cdot k}$	$\frac{R \cdot RTT_{min}}{n \cdot k'}$

and no buffering delay, which means that RTT is always equal to RTT_{min}.

Let R/n [pkt/s] represents the individual available bandwidth if n nodes fairly utilize the residual capacity and t_1 represent the time to fill the bottleneck link as shown in Fig. 1. Relationship between t_1 and R is expressed by

$$t_1 = \left(\frac{RTT_{min}}{x_{RTT}}\right) \cdot \left(\frac{R}{n} \cdot RTT_{min}\right) \tag{6}$$

where RTT_{min}/x_{RTT} is the time to increase the window size by one packet and $R/n * RTT_{min}$ is the number of packets needed to fill the residual capacity.

Table 2 shows t_1 calculated by using Table 1 and Eq. (6). This table demonstrates that t_1 of CR is constant irrespective of RTT. This means that flows with different RTT can utilize residual capacity fairly and contributes to RTT-fairness before packet buffering starts. However, t_1 of TCP-Reno and CI are variables of RTT and become longer as RTT increases. This means that shorter RTT flows dominate shared network resource against larger RTT flows when multiple flows compete.

Let us consider parameter setting of k and k' for CR and CI flows to behave similar to TCP-Reno having $RTT_{Reno,min}$. This is done by setting

$$k = \frac{1}{RTT_{Reno,min}^2} \tag{7}$$

$$k' = \frac{1}{RTT_{Reno,min}} \tag{8}$$

which result in the same t_1 as TCP-Reno.

3.2 Model II: Buffered Case

In Model II, packet buffering is carried out at the bottleneck router buffer until overflow. In this case, RTT becomes a time-varying variable of RTT round and is represented by $RTT = RTT_{min} + \alpha$, where α gives the number of buffered packets. Specifically, in case of TCP-Reno, RTT of the i-th RTT round is given by

$$RTT_i = RTT_{min} + \frac{i}{B} \tag{9}$$

where i of the right term provides the amount of buffered packets.

On the other hand, let $\Delta t_i'$ represent the time to increase the window size by one packet, which is given by

$$\Delta t_i' = \frac{RTT_i}{x_{RTT_i}} \tag{10}$$

where x_{RTT_i} denotes increment amount of the congestion window of the i-th RTT round. Then, let S/n [pkt] denote the available buffer size if n nodes fairly utilize buffer size at the bottleneck router and t_2 shows the total buffering time until overflow, which is given by

$$t_2 = \Delta t_1' + \Delta t_2' + \cdots + \Delta t_{S/n}' \tag{11}$$

Summation form of Eq. (11) is

$$t_2 = \sum_{i=1}^{S/n} \frac{RTT_i}{x_{RTT_i}} \tag{12}$$

where RTT in Table 1 is replaced to RTT_i, t_2 of the three protocols are calculated as shown in Table 3.

Table 3 Total buffering time of each protocol.

	TCP-Reno	CR	CI
t_2	$\sum_{i=1}^{S/n} RTT_i$	$\sum_{i=1}^{S/n} \frac{1}{k \cdot RTT_i}$	$\sum_{i=1}^{S/n} \frac{1}{k'}$

From this table, t_2 of CI does not include RTT_i so that total buffering time is always constant irrespective of RTT. This contributes to fair utilization of buffering space and RTT-fairness. However, other results suggest that TCP-Reno takes longer buffering time as RTT increases but CR takes shorter time until overflow.

Let us consider parameter setting of k and k' for CR and CI flows to behave the same as TCP-Reno having RTT_{Reno}, similar to the previous subsection. By combining Eq. (9) and Table 3, k and k' are given by

$$k_i = \frac{1}{RTT_{Reno}^2} = \frac{1}{\left(RTT_{Reno,min} + \frac{s}{B}\right)^2} \tag{13}$$

$$k_i' = \frac{1}{RTT_{Reno}} = \frac{1}{\left(RTT_{Reno,min} + \frac{s}{B}\right)} \tag{14}$$

where s [pkt] provides the amount of buffered packets, which is updated according to network condition. These settings result in the same t_2 as TCP-Reno.

4. Proposal

In this section, we present our proposal named "HRF (Hybrid RTT-Fair)-TCP" which achieves RTT-fairness along with throughput efficiency improvement and inter-protocol friendliness with TCP-Reno. This protocol requires only sender side modifications. Since we would like to satisfy fair utilization of common network resources (bottleneck capacity and buffer space), HRF-TCP has two phases in its window increment phase. Congestion window control of HRF-TCP is carried out as follows:

$$cwnd+ = k \cdot RTT^2 \quad [per\ RTT]$$
$$\text{when } RTT = RTT_{min} \tag{15}$$
$$cwnd+ = k' \cdot RTT \quad [per\ RTT]$$
$$\text{when } RTT > RTT_{min} \tag{16}$$
$$cwnd = max\left(\frac{RTT_{min}}{RTT} \cdot cwnd, \frac{cwnd}{2}\right)$$
$$\text{when a packet loss is detected} \tag{17}$$

where k and k' are parameters of RTT which are set according to Eq. (7) and Eq. (14), respectively. When vacant capacity exists (Model I, i.e. $RTT = RTT_{min}$), HRF-TCP increases its window size by CR algorithm. When buffering starts at the bottleneck router (Model II, i.e. $RTT > RTT_{min}$), it switches the mode to CI algorithm. Finally, when a packet loss happens, HRF-TCP decreases its window size like TCP-Westwood using Eq. (17) to keep throughput efficiency (i.e. clear the router buffer) and returns to CR or CI mode according to buffering status at the bottleneck router. Total behavior of HRF-TCP is shown in Fig. 2, where buffer size of the bottleneck router is less than BDP and HRF-TCP applies RTT_{min}/RTT for window decrement after packet losses.

5. Experiments

We carried out simulation experiments using ns-2 [16] and implementation experiments using Packet Storm [17].

Table 4 and Fig. 3 show the implementation environment and the experiment topology. There are n-flows, n-senders, n-receivers and two routers. All packets from senders and receivers are forwarded by two intermediate routers. This scenario is called "dumbbell topology". Sender i communicates with receiver i ($i = 1, 2, \ldots, n$). Each sender connects to the router with 1 Gbps link of which propagation delay is D_i which is varied according to RTT_i. Each receiver connects to the other router with 1 Gbps link of which propagation delay is 1 ms. Link speed and propagation delay of the shared (bottleneck) link between two routers are set to 100 Mbps and 1 ms, respectively.

We mainly use DT (Drop Tail) as the router buffer management algorithm. However, since DT is known to be vulnerable to the phase effect [4], as the solution, we show results of the random packet losses on the bottleneck link in all experiments in this paper.

In the experiments, TCP-Libra provides almost same results as CR so that we omit TCP-Libra by substituting CR. We set parameters k and k' of the CR/CI algorithms to the values calculated by assuming TCP-Reno of RTT = 40 [ms]. Automatic RTT estimation of competing flows is further study.

5.1 Model Verification

The first experiments are carried out to verify the window behavior model of Sect. 3. We observe single flow behaviors of RTT = 40 [ms] or RTT = 80 [ms]. According to parameter setting, bottleneck bandwidth (= B [pkt/s]) is given by

$$B = \frac{100 \cdot 10^6}{1500 \cdot 8} = 8333.3\dot{3}\ [\text{pkt/s}]$$

where packet size is 1500 [byte]. We set S (bottleneck router buffer size) to 500 [pkt] and compare window behaviors of TCP-Reno, CR, and CI.

Table 4 Machine specifications in the implementation. (All machines except the emulator are linux kernel 2.6.22)

Emulator	Sender	Receiver
Packet Storm 2600E	CPU:Pentium4 3GHz Memory:3GB	CPU:Pentium Dual E2180 2.0GHz Memory:2GB
	CPU:Pentium Dual E2180 2.0GHz Memory:2GB	CPU:Xeon 3040 1.86GHz Memory:1GB

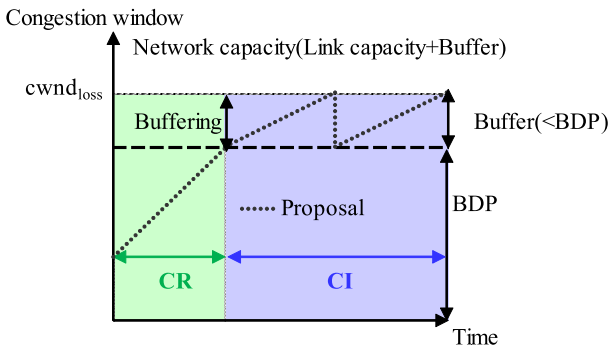


Fig. 2 HRF-TCP behavior.

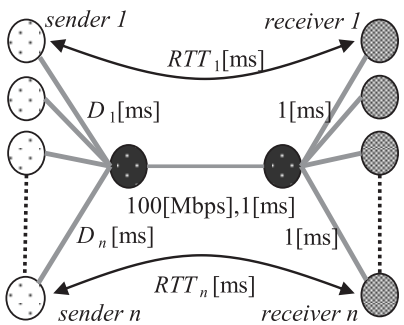


Fig. 3 Simulation topology.

5.1.1 Model I (Vacant Case)

We observe window behaviors until packets buffering starts at the bottleneck router. Assume R (residual capacity) is equal to $B/2$ [pkt/s], i.e. half utilization of the bottleneck link capacity, as a starting point of the simulation. In the end, window size reaches BDP and fills vacant capacity.

Figure 4 shows the window size behaviors obtained by Eq. (6) in our model and ns2 simulation. We add a vertical line corresponding to t_1 of RTT = 40 [ms], which is the time when the bottleneck link becomes fully utilized. As described before, k and k' in Eqs. (7) and (8) are calculated by assuming TCP-Reno of RTT = 40 [ms].

In Fig. 4, all protocols of RTT = 40 [ms] behave the same because of the parameter setting. t_1 of these flows are about 6.5 [s], which coincides with the estimated value from Table 2. However, window size behaviors of flows of RTT = 80 [ms] are different from each other. A most significant result is about TCP-Reno, in which t_1 is approximately 4 times as many as that of RTT = 40 [ms]. Since CI increases its window size in the same rate of the RTT = 40 [ms] flow, it needs twice the time to fill vacant capacity compared with the RTT = 40 [ms] flow. Finally, t_1 of CR is the same as the 40 [ms] case as we estimated in Sect. 3.1. When we compare our model with simulation, all the flows show similar behaviors. This validates Model I.

5.1.2 Model II (Buffered Case)

Congestion window size behaviors from buffering start to overflow are shown in Fig. 5. The result shows ns2 simulation and analytical estimation by our model using Eq. (12). Window size right before overflow is $BDP + S$ [pkt]. Parameters, k and k', are calculated by Eq. (13) and Eq. (14), respectively, with $RTT_{Reno,min}$ = 40 [ms]. A vertical line means t_2 of the RTT = 40 [ms] case which represents the time from buffering start to overflow.

In Fig. 5, congestion window size behaviors show curved lines which mean that increment rates become slow because of the extra delay caused by packet buffering. Although all protocol flows of RTT = 40 [ms] behave the same, flows of RTT = 80 [ms] perform differently. A TCP-Reno flow slowly fills up buffer space but CR rapidly fills. In case of CI, increment rates are the same and t_2 is constant regardless of RTT values. Both our model and simulation results show almost the same behaviors. Then, validity of Model II is also proved.

Though we omit here, implementation experiments also brings similar results in Model I and Model II. From the verification results so far, we can conclude that CR fairly utilizes residual capacity and CI fairly utilizes buffer spaces.

5.2 Intra-Protocol RTT-Fairness

We inspect intra-protocol RTT-fairness in this subsection. We define a notation for combined protocol usages by $Dec + M1 + M2$, in which Dec means window decrement phase upon packet losses, $M1$ means window increment phase when bottleneck link is vacant (Model I), and $M2$ means window increment phase when packets are buffered (Model II). For Dec, we consider TCP-Reno (halving congestion window) and TCP-Westwood (clearing buffer), and denote them by "Reno" and "W", respectively. For $M1$ and $M2$, we consider Reno, CR and CI, and denote them as they are. Then, we compare next protocols:

- Reno
- CR
- CI
- W+CR+Reno
- W+CR+CR
- HRF (=W+CR+CI)

W+CR+Reno is an algorithm which uses CR algorithm when there is residual capacity on the bottleneck link and behaves as Reno after the link is fully utilized. Similar to this, W+CR+CR is an algorithm which uses CR in whole window increment phase. HRF is an intermediate between W+CR+Reno and W+CR+CR, and it can be denoted by W+CR+CI.

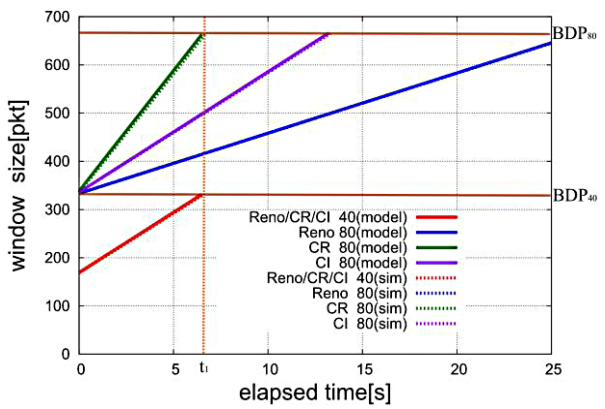


Fig. 4 Congestion window behaviors obtained by Eq. (6) and simulation for Model I.

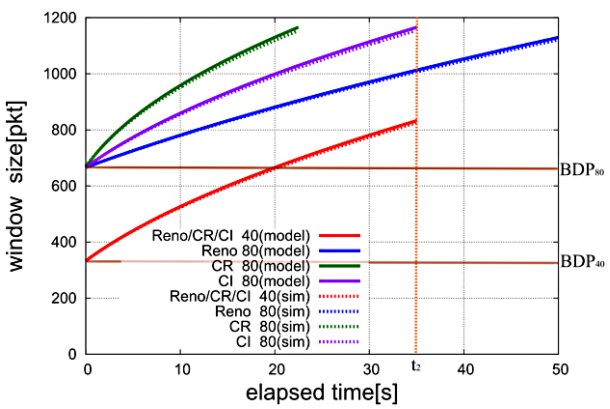


Fig. 5 Congestion window behaviors obtained by Eq. (12) and simulation for Model II.

In the experiments, two flows of the same protocol but having different RTTs run in the topology in Fig. 3. One flow has RTT = 40 [ms] (BDP = 333 [pkt]) and the other flow has RTT = 120 [ms] (BDP = 1000 [pkt]). From Fig. 6 to Fig. 8, figures show throughput differences, Thr1-Thr2, where Thr1 is a throughput of the RTT = 40 [ms] flow and Thr2 is that of the RTT = 120 [ms] flow.

5.2.1 Changing Buffer Size

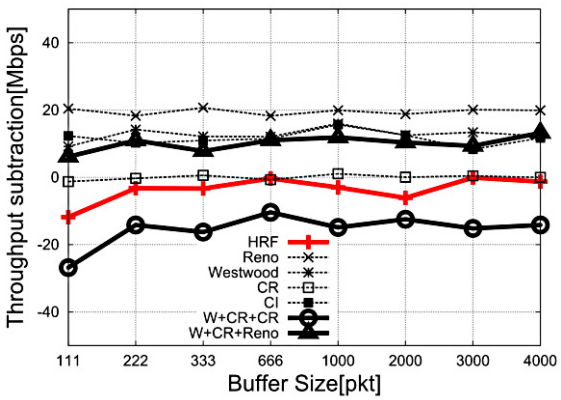
We evaluate intra-protocol RTT-fairness when changing buffer size at the bottleneck router. Buffering size at the bottleneck router is variable between 111∼4000 [pkt].

From Fig. 6, we can recognize that HRF and CR present the best RTT-fair result. In the non-buffered state (Model I), CR is a more RTT-fair contributor than CI and Reno. In case of CI and Reno, the smaller RTT flow gets more resources than the longer RTT flow. In the buffered state (Model II), W+CR+CR provides more resources for longer RTT flow but W+CR+Reno does for smaller RTT flow. The results prove that the CI behavior of HRF contributes to better RTT-fairness. Using "W" instead of "Reno" in window decrement phase cause different effects to RTT-fair according to window increment phase behavior, better (compare Reno with Westwood) or worse (compare CR with W+CR+CR). We also note that adaptive switching between CR and CI brings much better RTT-fairness performance than the others when TCP-Westwood is used as the decrement phase.

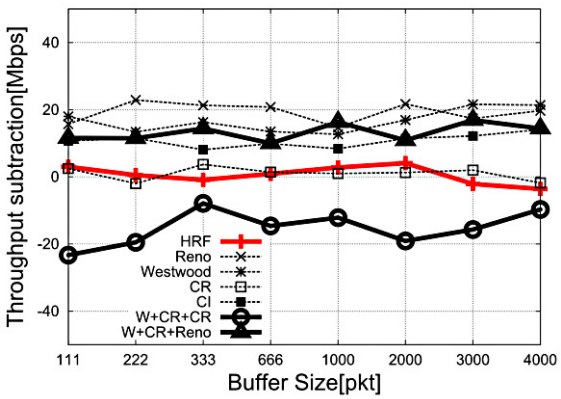
5.2.2 Changing RTT Value

To evaluate intra-protocol RTT-fairness by changing RTT value, we set one flow to have constant RTT (= 40 [ms]) and the other flow to have variable RTT (= 13.33∼480 [ms]). Buffer size of the bottleneck router is fixed at 333 [pkt].

In Fig. 7, HRF and CR are almost RTT-fair regardless of RTT values. In Model I, by comparisons of three protocols (Reno, CI and CR), same trends as the previous section are shown. CI and Reno get RTT-unfair as a RTT difference between two flows becomes large. In Model II, by comparisons of three protocols (W+CR+Reno, HRF and W+CR+CR), CI improves RTT-fair regardless of RTT values though CR and Reno negatively affect RTT-fairness. Differently from the previous section, applying "W" into Reno and CR make RTT-fair worse as a RTT difference be-

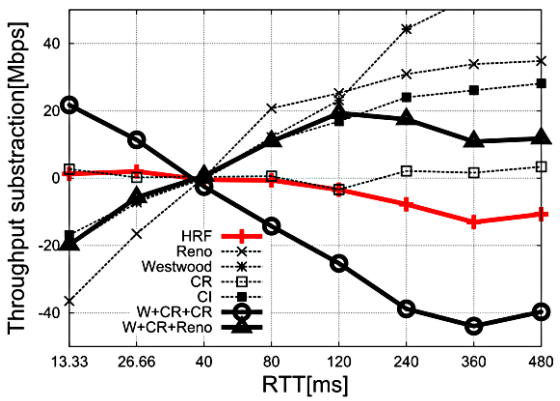


(a) Simulation

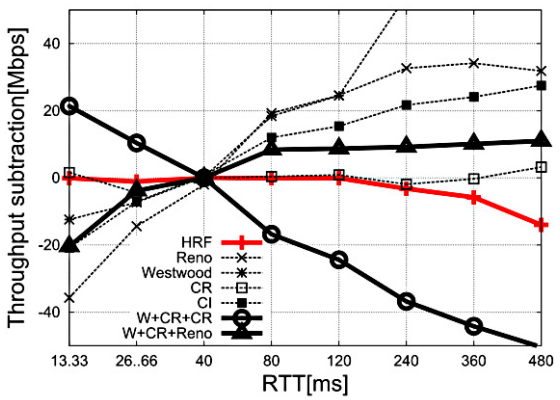


(b) Implementation

Fig. 6 Throughput differences when changing buffer size.



(a) Simulation



(b) Implementation

Fig. 7 Throughput differences when changing RTT value.

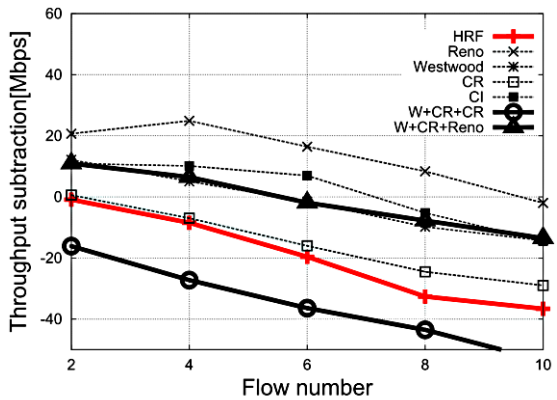
tween two flows becomes large. We conclude that HRF gets RTT-fairness without changing RTT value despite of using "W" in window decrement phase.

5.2.3 Changing the Number of Flows

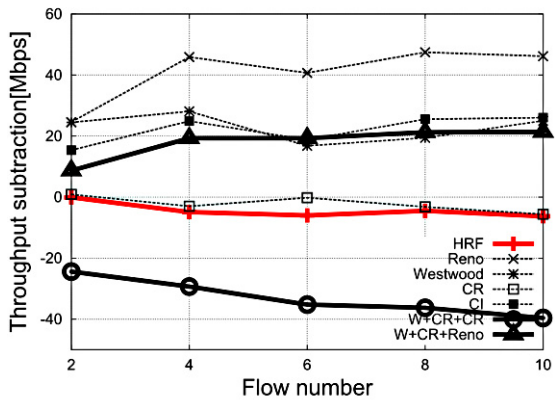
We carry out the experiments by changing the number of flows of the same protocol. RTTs of half flows are 40 [ms] and those of the others are 120 [ms]. Buffer size of bottleneck is 333 [pkt]. There is a difference between simulation and implementation. Due to hardware limitation of our network emulator which has two incoming interfaces and two outgoing interfaces, each node generates multiple TCP flows and multiplex them on the access link.

Figure 8 shows that larger RTT flows of all protocols tend to utilize more bandwidth as the number of flows increase in the simulation case. This is because one flow tends to dominate the bandwidth and does not provide any chances to increase window sizes of the other flows. By contrast, in the implementation experiments, Fig. 8 (b) shows relatively stable results regardless the number of flows. The trends in the term of RTT-fairness are similar to 5.2.1 section. Reno, CI and W+CR+Reno flows with smaller RTT dominantly utilize bandwidth, but W+CR+CR behaves oppositely. Note that HRF and CR are RTT-fair as we expected in spite of changing the number of competing flows.

In conclusion, HRF, which combines Westwood, CR and CI algorithm in each phase, brings much better RTT-fair in all cases changing buffer size, RTT value, and the number of flows.



(a) Simulation



(b) Implementation

Fig. 8 Throughput differences when changing the number of flows.

5.3 Throughput Efficiency

Figure 9 compares throughputs of a single TCP flow when RTT is varied from 13.33 [ms] to 480 [ms] and random packet losses are caused by 10^{-6}. Buffer size at the bottleneck router is constant at 100 [pkt], which is smaller than BDP and tends to cause vacant capacity in the bottleneck link. In Fig. 9, continuous lines and dot lines are implementation and simulation results, respectively.

From this figure, we can recognize that HRF can bring sufficient throughputs but the others cannot. TCP-Westwood performs well for small RTTs but degrades for larger RTTs because of its conservative window increment. TCP-Reno and CI performs worse as RTTs become larger due to their window halving upon packet losses. CR also decreases its throughputs as RTTs increases because, in these cases, almost packet losses are caused by overflow and vacant capacity is increased. Even in such these cases, only HRF performs well thanks to its RTT-adaptive window increment and TCP-Westwood's smart window decrement. When RTT value becomes 480 ms, the throughputs of HRF and CR in the simulation degrade far from those in the implementation. The reason is that increment amount of window size per RTT by CR algorithm exceeds the buffer size, which lead to the burst packet losses and timeout happens. However, we reason that some factors (delay jitter) would mitigate its effect in the implementation.

5.4 Inter-Protocol RTT Fairness

This subsection focuses on inter-protocol RTT fairness with TCP-Reno. Buffer size is equal to BDP. Packet loss rate is set to 10^{-4} to eliminate the phase effect. Figure 10

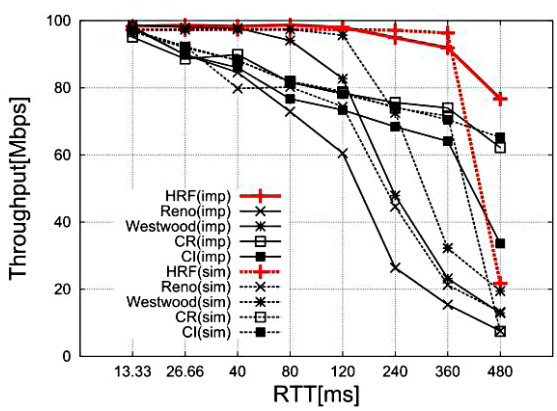


Fig. 9 Throughputs of various TCPs when RTT varies.

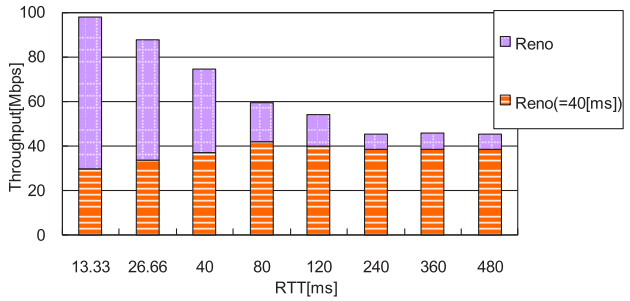


Fig. 10 Throughputs of competing TCP-Reno flows having different RTTs.

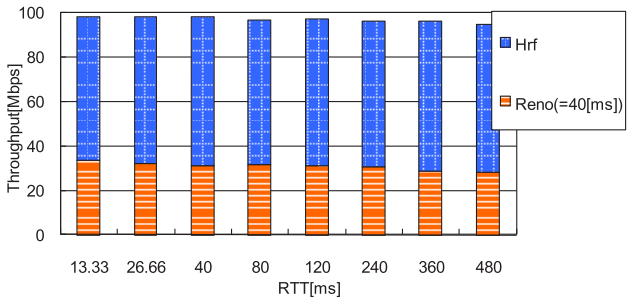


Fig. 11 Throughputs of competing TCP-Reno and HRF flows having different RTTs.

shows throughputs of competing two TCP-Reno flows, and Fig. 11 shows those of TCP-Reno and HRF flows. These results come from implementation. Simulation results are almost same as them. One TCP-Reno flow has constant RTT (40 [ms]) and the other flow (TCP-Reno or HRF) has different RTT varied from 13.33 [ms] to 480 [ms].

From these figures, we can observe that total throughput of the HRF/Reno case is significantly larger than that of the TCP-Reno only case. This is because HRF is an efficient protocol as proved in the previous subsection. Figure 10 shows RTT-unfairness of TCP-Reno, which had been indicated by many researchers. On the other hand, Fig. 11 shows stable throughputs of TCP-Reno and HRF in spite of different RTTs. This indicates that HRF can utilize bandwidth efficiently without disturbing TCP-Reno, and achieve inter-protocol RTT-fairness with TCP-Reno even if RTT changes. HRF keeps friendliness with TCP-Reno by appropriate k and k' settings, which control proposal of tradeoff between scalability and friendliness, though, we should take an investigation into those parameters.

6. Conclusions

In this paper, we focused on resource sharing (of the bottleneck link and the router buffer) and analyzed the unfairness problems of them. By the different approach from the classical method named CR, we then presented a new congestion control algorithm called "HRF-TCP" to support RTT-fairness which switches CR and CI modes according to observed delays and applies TCP-Westwood's window decrement mechanism. Both implementation and simulation results proved that HRF provides RTT-fairness, throughput efficiency and TCP-friendliness with TCP-Reno. As future work, we should determine k and k' automatically according to the RTT estimation of competing flows.

References

[1] W.R. Stevens, "TCP slow start, congestion avoidance, fastretransmit, and fast recovery algorithms," IETF RFC 2581, 1997.
[2] H. Hisamatu, H. Ohsaki, and M. Murata, "Steady state analysis of TCP connections with different propagation delays," IEICE Technical Report, IN2002-97, Oct. 2002 (in Japanese).
[3] G. Marfia, C.E. Palazzi, G. Pau, M. Gerla, M.Y. Sanadidi, and M. Roccetti, "Balancing video on demand flows over links with heterogeneous delays," ACM MobiMedia 2007, Aug. 2007.
[4] S. Floyd and V. Jacobson, "On traffic phase effects in packet-switched gateways," ACM SIGCOMM Computer Communication Review, vol.21, no.2, pp.26–42, 1991.
[5] T.H. Henderson, E. Sahouria, S. McCanne, and R.H. Katz, "On improving the fairness of TCP congestion avoidance," IEEE Globecomm, 1998.
[6] L. Xu, K. Harfoush, and I. Rhee, "Binary increase congestion control (BIC) for fast, long distance networks," Proc. INFOCOM, 2004.
[7] S. Ha, I. Rhee, and L. Xu, "CUBIC: A new TCP-friendly High-speed TCP variant," ACM SIGOPS Operating Systems Review, vol.42, Issue 5, pp.64–74, July 2008.
[8] C. Jin, D.X. Wei, and S.H. Low, "FAST TCP: Motivation, architecture, algorithms, performance," IEEE INFOCOM 2004, March 2004.
[9] Y. Nemoto, K. Ogura, and J. Katto, "TCP congestion control using RTT estimation by measuring ACK intervals," IEICE Technical Report, NS2010-270, March 2011 (in Japanese).
[10] C. Casetti, M. Gerla, S. Mascolo, M.Y. Sanadidi, and R. Wang, "TCP westwood: Bandwidth estimation for enhanced transport over wireless links," Proc. ACM Mobicom 2001, July 2001.
[11] K. Tan. J.Song, Q. Zhang, and M. Sridharan, "Compound TCP: A scalable and TCP-friendly congestion control for high-speed networks," PFLDnet 2006, Feb. 2006.
[12] H. Shimonishi, T. Hama, and T. Murase, "TCP-adaptive reno for improving efficiency-friendliness tradeoffs of TCP congestion control algorithm," PFLDnet 2006, Feb. 2006.
[13] S. Liu, T. Başar, and R. Srikant, "TCP-Illinois: A loss and delay-based congestion control algorithm for high-speed networks," VALUETOOLS 2006, Oct. 2006.
[14] A. Baiocchi, A.P. Castellani, and F. Vacirca, "YeAH-TCP: Yet another highspeed TCP," PFLDnet 2007, Feb. 2007.
[15] K. Kaneko, T. Fujikawa, S. Zhou, and J. Katto, "TCP-fusion: A hybrid congestion control algorithm for High-speed networks," PFLDnet 2007, Feb. 2007.
[16] "ns-2 network simulator(ver.2)," http://www.mash.cs.berkley.edu/
[17] Packet Storm, http://www.packetstorm.com

Kazumine Ogura received the B.Eng. degree in computer engineering from Waseda University, Tokyo, Japan in 2008, and the M.Eng. degree in computer engineering from Waseda University, Tokyo, Japan in 2009. His current research interests include the TCP, wired and wireless networks. He is currently working towards the Ph.D. degree in the Graduate School of Science and Engineering, Waseda University.

Yohei Nemoto received the B.Eng. degree in computer engineering from Waseda University, Tokyo, Japan in 2008. His current research interests include the TCP, wired and wireless networks. He is currently working towards the M.Eng. degree in the Graduate School of Science and Engineering, Waseda University.

Zhou Su received the B.E. and M.E. degrees from Xi'an Jiaotong University, Xi'an, China, in 1997, 2000, and Ph.D. degree from Waseda University, Tokyo, Japan, in 2003, respectively. He was an exchange student between Waseda and Xi'an Jiaotong University from 1999 to 2000. From 2001 he had been a research associate at Waseda University and he is currently an assistant professor at the same university. His research interests include multimedia communication, web performance and network traffic. He received the SIEMENS Prize in 1998, and ROCKWELL Automation Master of Science Award in 1999. He is a member of the IEEE and IEE.

Jiro Katto received B.S., M.E. and Ph.D. degrees in electrical engineering from University of Tokyo in 1987, 1989 and 1992, respectively. He worked for NEC Coporation from 1992 to 1999. He also a visiting scholar at Princeton University, NJ, USA, from 1996 to 1997. He then joined Waseda University in 1999. Since 2004, he has been a professor of the Department of Computer Science, Science and Engineering, Waseda University. From 2004 to 2008, he has also been a director of the Electronic and Information Technology Development Department, New Energy and Industrial Technology Development Organization. His research interest is in the field of multimedia communication systems and multimedia signal processing. He received the Best Student Paper Award at SPIE Conference of Visual Communication and Image Processing in 1991, and received the Young Investigator Award of IEICE in 1995. He is a member of IEEE, ACM, IPSJ and ITE.