TCP-Fusion: A Hybrid Congestion Control Algorithm for High-speed Networks

Kazumi KANEKO, Tomoki FUJIKAWA, Zhou SU and Jiro KATTO

Graduate School of Science and Engineering, Waseda University
3-4-1 Okubo, Shinjyuku-ku, Tokyo, 169-8555 Japan
E-mail: {kaneko, katto}@katto.comm.waseda.ac.jp

***Abstract*— This paper presents a new congestion control algorithm of TCP, called TCP-Fusion, and provides its extensive evaluation results through simulations and implementations. Recently, towards high-speed networks with large bandwidth delay product, a number of different approaches have been proposed to improve TCP performance. However, their potential unfriendliness to TCP-Reno encumbers their wide deployment in the Internet because TCP-Reno is already widely deployed. Most recently, to satisfy efficiency and friendliness tradeoffs of TCP, new approaches combining a loss-based protocol and a delay-based protocol have been proposed, such as TCP-Adaptive Reno and Compound TCP. Our TCP-Fusion also belongs to this category and tries to utilize the residual capacity effectively without impacts on coexisting flows, i.e. TCP-Reno flows. To achieve this purpose, TCP-Fusion exploits three useful characteristics of TCP-Reno, TCP-Vegas and TCP-Westwood in its congestion avoidance strategy. In short, congestion window of TCP-Fusion is decreased without causing too drastic reduction and is increased with smart adaptability to coexisting TCP-Reno flows according to the congestion level measurement estimated from RTT. Our implementation and simulation results show that TCP-Fusion can obtain the highest throughput among existing TCP variants when there is unused residual capacity while its friendliness to the TCP-Reno is sufficiently satisfied, otherwise, it shares the same bandwidth to coexisting flows.**

I. INTRODUCTION

TCP (Transmission Control Protocol) is widely used in current network, provides end-to-end, reliable congestion control. The majority of data services including FTP and HTTP in the Internet are carried by TCP. Recently, while the amount of Internet traffic is explosively increasing with the rapid growth of Internet users, the Internet is evolving to high-speed networks with large bandwidth delay product (BDP). However it is well known that current TCP (mostly TCP-Reno [1] and TCP-NewReno [2]) throughput deteriorates in such high-speed networks. This is believed to be primary due to the congestion control algorithm of TCP-Reno, whose congestion window size is increased by 1*MSS/RTT* and halved upon packet losses regardless of network condition. That is to say, its window control mechanism is too conservative in increase and drastic in decrease for high-speed networks, respectively.

To overcome this issue, a number of different approaches have been proposed. These approaches can be classified into three categories. One modifies an *AIMD* (Additive Increase Multiplicative Decrease) mechanism of TCP congestion avoidance phase to quickly increase and slowly decrease the congestion window than TCP-Reno, to achieve high throughput in high-speed networks. This approach is called a *loss-based protocol*, which adjusts its congestion window size by causing packet losses intentionally. Examples are High-speed TCP (HSTCP, for short) [3], Scalable TCP [4], early version of TCP-Westwood (TCPW) [5, 6], BIC [14] and CUBIC [16]. In contrast to these loss-based protocols, as the second category, a *delay-based protocol* makes use of the RTT as a network congestion estimator and can achieve excellent steady state performance. Examples are TCP-Vegas [7] and FAST TCP [8]. These advanced protocols promise to improve the TCP performance significantly in high-speed networks. As many researchers were pointing out, however, their potential unfriendliness to TCP-Reno stands in the way of their wide deployment. Since TCP-Reno is already widely used, not only improving TCP performance but also friendliness to TCP-Reno is one of the most important issues in designing a new protocol. Recently, to manage efficiency and friendliness tradeoffs above, as the third category, *loss-based protocols using RTT metrics* have been proposed, e.g., Gentle high-speed TCP [11], TCP-Africa [12], TCP-Adaptive Reno (ARENO) [9] and Compound TCP (CTCP) [13]. They can adaptively switch their congestion control mode or TCP response function according to the congestion level measurement estimated from RTT.

In this paper, we propose a new hybrid congestion control algorithm of TCP. This new protocol, called TCP-Fusion, makes use of three useful characteristics of TCP-Reno, TCP-Vegas and TCPW, and also belongs to the *loss-based protocol using RTT metric* category mentioned above. The key concept is similar to the existing ones, whose congestion window sizes are increased aggressively whenever the network is estimated underutilized. When the network is fully utilized, they perform similarly to the TCP-Reno's manner. Our proposal, by modifying these characteristics to be a scalable manner, can adjust its congestion window according to the left unused capacity, and provides a good balance among high efficiency and friendliness to TCP-Reno. Our implementation and simulation evaluations including other TCP variants show the effectiveness of TCP-Fusion.

This paper is organized as follows. In section Ⅱ, we describe the TCP-Fusion algorithm. Section Ⅲ provides the implementation and simulation results. Finally we conclude this paper.

II. TCP-FUSION PROTOCOL

TCP-Fusion maintains two congestion window sizes. The one has two properties of TCP-Vegas and TCPW, which provide efficiency in large leaky pipe, i.e., the link with large bandwidth-delay product and non-negligible random losses. The other one is updated like TCP-Reno, i.e., the value is increased by 1*MSS/RTT* and halved upon packet losses, and then TCP-Fusion adopts either big one as its new congestion window size. Therefore, TCP-Fusion ensures at least TCP-Reno performance.

A. Congestion Window Reduction

TCP-Fusion adopts optimization of the decrease parameter based on TCPW-RE (Rate Estimation) [6] to improve efficiency particularly in the leaky pipe. In TCPW-RE, the decrease parameter after a loss can be expressed as RTT_{min}/RTT [10], where RTT_{min} and RTT are the minimum RTT and the RTT right before the packet loss, respectively. This equation indicates that TCPW-RE reduces its congestion window size to clear the buffer and, as a result, it is friendly to TCP-Reno only if the buffer capacity is equal to the *BDP* [10, 17] where RTT grows up to $2*RTT_{min}$. If the buffer capacity is larger than the *BDP*, the decrease value is less than 1/2 upon a congestion loss (Fig, 1) and TCPW-RE cannot obtain a share of the bandwidth. To address this issue, we then set the thresholds to 1/2 as follows;

$$cwnd_{new} = \max(\frac{RTT_{\min}}{RTT} cwnd_{last}, \frac{cwnd_{last}}{2})$$

where $cwnd_{new}$, and $cwnd_{last}$ are congestion window sizes right after and before the packet loss, respectively.

B. Congestion Window Increase

Similar to TCP-Vegas, TCP-Fusion has three phases; increase phase, decrease phase, and steady phase, which are switched by a number of packets in the bottleneck queue (*diff*). The *diff* can be estimated as;

$$diff = cwnd \frac{(RTT - RTT_{\min})}{RTT}$$

If the *diff* is less than the *lower bound threshold*, the link is determined as underutilized, and its congestion window size is increased rapidly to fill the pipe size. If the *diff* is larger than the *upper bound threshold*, the link is determined as utilized and early congestion, and its congestion window size is decreased to the value that has at least the *lower bound threshold* in the bottleneck queue. Otherwise, the link is determined in a good balance and non-congestion condition, and its congestion window size is fixed. Fig. 2 shows the congestion window behavior of TCP-Fusion when TCP-Fusion and TCP-Reno are competing with a buffer size less than BDP.

$$cwnd_{new} = \begin{cases} cwnd_{last} + W_{inc} / cwnd_{last}, & \text{if } diff < \alpha \\ cwnd_{last} + (-diff + \alpha) / cwnd_{last}, & \text{if } diff > 3*\alpha \\ cwnd_{last}, & \text{otherwise} \end{cases}$$

$$\text{cwnd}_{\text{new}} = reno\_cwnd, \quad \text{if } cwnd_{new} < reno\_cwnd$$

where $cwnd_{new}$, $cwnd_{last}$ and $reno\_cwnd$ are the congestion window sizes after and before update and of an equivalent to TCP-Reno, respectively. α is the *lower bound threshold* to switch three phases. W_{inc} is the increment parameter to increase congestion window size rapidly. With regard to α and W_{inc}, we will discuss in detail in the next subsection.

C. Setting Parameters

Setting the threshold parameter α has a big impact on the performance of our proposal. We first consider next two requirements for the parameter setting:

1) Considering friendliness to TCP-Reno, it should be a small value to minimize the queuing delay that affects coexisting TCP-Reno flow. However, if α is too small compared to the link bandwidth, it becomes meaningless due to TCP timer granularity (*diff* is always larger than α except the case of $RTT=RTT_{min}$). As a result, three phases of TCP-Fusion cannot be switched adequately. Therefore α should be proportional to the link bandwidth.

2) When there are coexisting N TCP-Fusion flows, since each flow will try to put α packets into the bottleneck buffer, the router buffer size has to accommodate at least $N*\alpha$ packets. If the router buffer size is less than the value, the parameter *diff* never goes up to α. This means that all TCP-Fusion flows always increase their congestion window sizes aggressively, and results in unneeded frequent buffer overflows. Therefore, α should be set to a small value according to the number of coexisting TCP-Fusion flows.

To satisfy these requirements mentioned above, we start with an assumption that no routers have smaller than G packets that corresponds to the queuing delay D_{min} in the bottleneck queue. Thus, G is given by

$$G = \frac{B * D_{\min}}{8 * packet\_Size}$$

where B is the bandwidth of bottleneck link. In the worst case that all coexisting N flows employ TCP-Fusion algorithm, since the total packets ($N*\alpha$) is equal to G packets, α can be expressed by G/N. Although it is hard to know the bottleneck link bandwidth B and the coexisting number of flows N accurately,

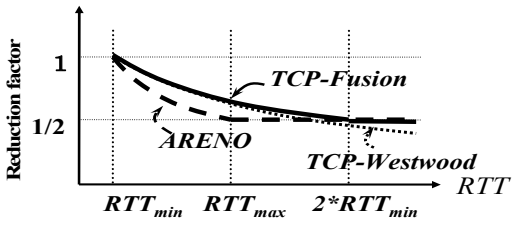

Fig. 1: Reduction factor

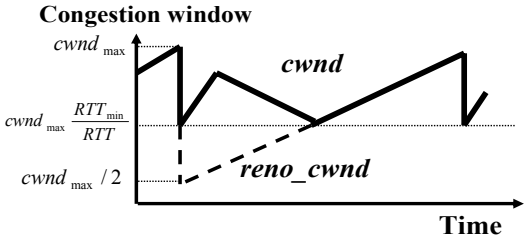

Fig. 2: Congestion window behavior

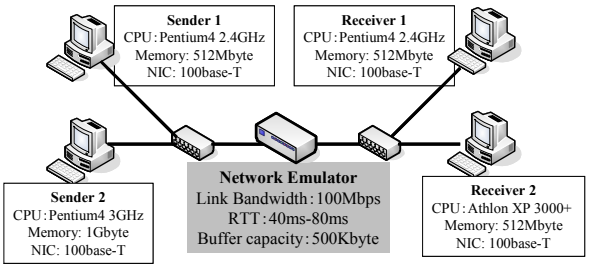

Fig. 3: Network model in laboratory

we can approximate the value B/N by achieved rate estimation as follows:

$$\alpha = \frac{G}{N} = \frac{(B/N)*D_{\min}}{packet\_size*8} \approx \frac{RE*D_{\min}}{packet\_size*8}$$

where RE is the achieved rate estimation governed by TCPW-RE (Rate Estimation) scheme. By setting this value, α scales up to the link capacity (Requirement 1), and becomes a small value in inverse proportion to the number of coexisting TCP-Fusion flows (Requirement 2). Furthermore, since the achieved rate RE is equivalent to $cwnd/RTT$, α can be expressed by

$$\alpha = \frac{RE*D_{\min}}{packet\_size*8} = cwnd\frac{D_{\min}}{RTT}$$

This equation means that accuracy of the parameter α is limited by the TCP timer granularity (*tcp_tick*) of which value depends on operating system. We can set D_{min}= *tcp_tick* or higher value to absorb slight oscillations of the RTTs in dynamic network conditions. In this paper, while our simulation experiments showed that D_{min}=*tcp_tick* is controllable enough, we set it to 4ms based on our implementation experiments as of now.

Secondly, we set W_{inc} according to our assumption that no routers have smaller than G packets. In TCP-Fusion, even if its congestion window size reaches the *BDP*, since RTT doesn't change from RTT_{min} yet and thus the network is recognized as underutilized, the congestion window size is increased by W_{inc}. Accordingly, if W_{inc} is set to more than G packets, packet losses are caused at the time on our assumption. Thus, W_{inc} is upper bounded by

$$W_{inc} \le G = \frac{B*D_{\min}}{8*packet\_size}$$

where B is the bandwidth estimation achieved by time sequence of ACKed sequence numbers like TCPW-BE. By this way, its increment parameter can scale up to the link bandwidth. While we can set D_{min} to 4ms as with α mentioned above, our simulation experiments show that D_{min}=1*ms* is reasonable value even in high-speed networks. When we assume D_{min}=1*ms* and 1500*B* packet size, we have W_{inc}=*B*/12*Mbps*.

III. Performance Evaluations

We carried out extensive simulations using ns-2 simulator [18] and implementation experiments on a Linux system to verify TCP-Fusion properties. In our implementation experiments, we newly implemented TCP-Fusion, ARENO and CTCP as Linux modules (Kernel 2.6.15), and tested six TCP variants; TCP-Fusion, ARENO, CTCP, HSTCP, BIC and CUBIC. In this paper, we mainly introduce implementation results inside our laboratory and on an actual network experiments. We omit a lot of simulation results which support implementation results in this paper due to page limitation. Instead, we only show a few simulation results that cannot be evaluated in our current implementation environments. The parameter β is set to 12Mbps.

A. Laboratory Experiments

The network topology is shown in Fig. 3. The bandwidth, round-trip propagation delay and buffer size are illustrated in this figure. The 500KB of buffer size is equal to the BDP of setting 40ms RTT. For a traffic source, we use *Iperf*[19] to generate continuous TCP data flow.

1) Efficiency and Friendliness in lossy link

Fig. 4 shows the throughput of a single TCP flow. For the network emulator setting, RTT is 40ms and random packet loss rate is varied from 10^{-1} to 10^{-6}. All kinds of TCP variant flows can utilize nearly the link bandwidth when the loss rate is smaller than 10^{-5} and degrade its throughput as the loss rate increases. Among of them, TCP-Fusion is most efficient and robust in this lossy link. For example, TCP-Fusion can obtain

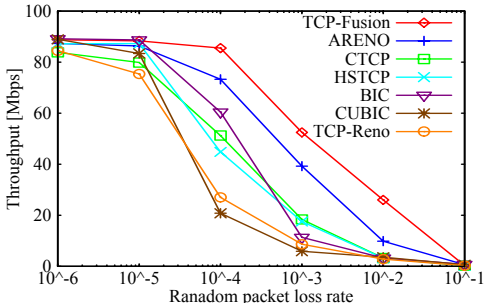

Fig. 4: Throughput of a single flow with different loss rate

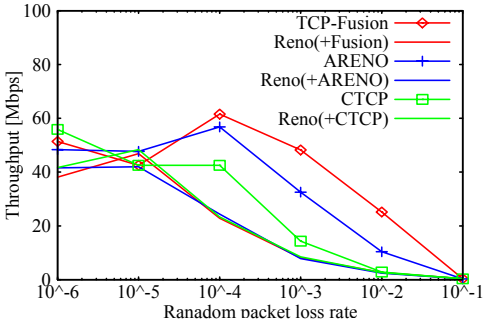

Fig. 5: Throughput of coexisting two flows with different loss rate (the cases of Fusion/ARENO/CTCP)

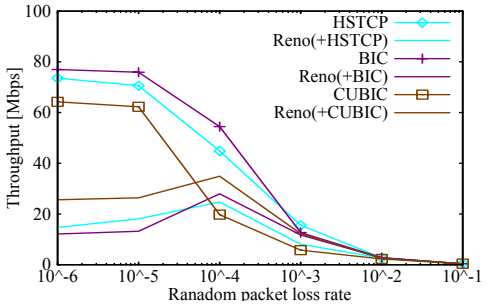

Fig. 6: Throughput of coexisting two flows with different loss rate (the cases of HSTCP/BIC/CUBIC)

Table 1: Throughput of coexisting two TCP variant flows

	Reno	Fusion	ARENO	CTCP	HSTCP	BIC	CUBIC
Reno	44.9	40.2	42.4	38.6	10.3	10.4	18.6
Fusion	50.0	43.6	46.0	43.0	21.1	11.4	18.7
ARENO	40.5	44.0	43.2	38.0	16.5	12.2	19.4
CTCP	53.9	46.5	50.5	44.9	20.8	13.8	15.0
HSTCP	78.2	61.4	68.2	65.8	43.2	29.9	29.7
BIC	81.2	79.0	78.0	76.2	59.7	44.8	45.4
CUBIC	71.7	70.3	68.4	74.0	59.6	43.9	44.3

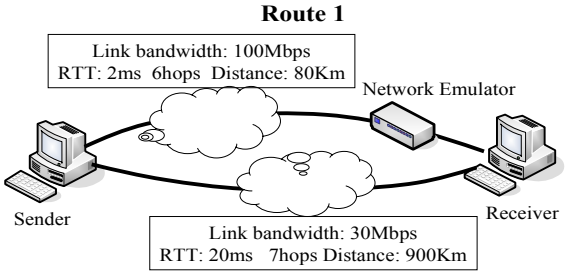

Fig. 7: Actual network lines

up to 1.4 times of ARENO throughput in Fig. 4. This is because RTT_{max} of ARENO, as the loss rate increases, becomes less than $2*RTT_{min}$ due to reduction of congestion losses. In such cases, the reduction factor upon a packet loss of ARENO is less than those of TCPW or TCP-Fusion as shown in Fig. 1.

We then evaluate the friendliness in lossy link. In this experiment, TCP variants and TCP-Reno flows are competing. Fig. 5 and Fig. 6 present the results of TCP-Fusion/ARENO/ CTCP and HSTCP/BIC/CUBIC, respectively. If the loss rate is smaller than 10^{-5}, TCP-Reno flow can obtain fair share (50Mbps) of the link bandwidth. Therefore, TCP variant flow should have the same bandwidth to coexisting TCP-Reno flow until the loss rate is 10^{-5}, and after that, utilizes the residual capacity left unused by the TCP-Reno flow. In Fig. 5, we can see that TCP-Fusion, ARENO and CTCP flows have approximately the same bandwidth to the TCP-Reno flow until the loss rate is 10^{-5}. After that, when the loss rate increases, they can obtain more than fair share by utilizing the residual capacity. In this experiment again, TCP-Fusion can obtain the highest throughput. On the other hand, HSTCP, BIC and CUBIC deteriorate coexisting TCP-Reno flow throughput to much less than fair share when the loss rate is small in Fig. 6.

2) Friendliness among TCP variants

In this experiment, we evaluate the friendliness when two TCP variant flows are competing. Previously, the target for evaluation of the friendliness is argued only for TCP-Reno. However, on the present situation where BIC is implemented in Linux by default and CTCP will be implemented in Windows Vista, we anticipate several TCP variants will be competing in the future Internet. Accordingly, for TCP variants, the friendliness to other TCP variants as well as TCP-Reno becomes one of the important concerns.

The experimental result is shown in Table 1. This table means that, for example, when TCP-Fusion and ARENO are competing, TCP-Fusion flow throughput is 46.0Mbps and ARENO is 44.0Mbps. We can see through this experiment that TCP variants with high friendliness such as TCP-Fusion, ARENO and CTCP have almost the same bandwidth without critical impacts with each other. By contrast, BIC and CUBIC flows deteriorate the other TCP variants and even HSTCP throughput. TCP variants flows coexisting with BIC or CUBIC flow have no chance to increase its congestion window quickly because BIC or CUBIC flow grows its congestion window regardless of its available capacity.

B. Actual Network Experiments

We prepared two actual network lines as shown in Fig. 7; Tokyo-Honjyo line (*Route* 1) and Tokyo-Kitakyushu line (*Route* 2). These two lines are the part of backbone networks of Waseda University. Their link characteristics are illustrated in this figure. To avoid the influence of unexpected background traffic, the throughput presented below is the average of several trials.

1) Fairness among identical flows in Route 1

We first evaluate the fairness when three identical TCP variant flows are competing. The additional RTT at network emulator is set to 80ms. No additional packet loss rate is introduced. The first, second and third flows start at 0s, 50s and 100s, respectively. Fig. 8 shows the congestion window behavior of each TCP variant case. The results show that all kinds of TCP variants except HSTCP behave almost fairly. HSTCP shows slow convergence with heavy oscillation. Their throughputs of the first, second and third flows are 51.07Mbps,

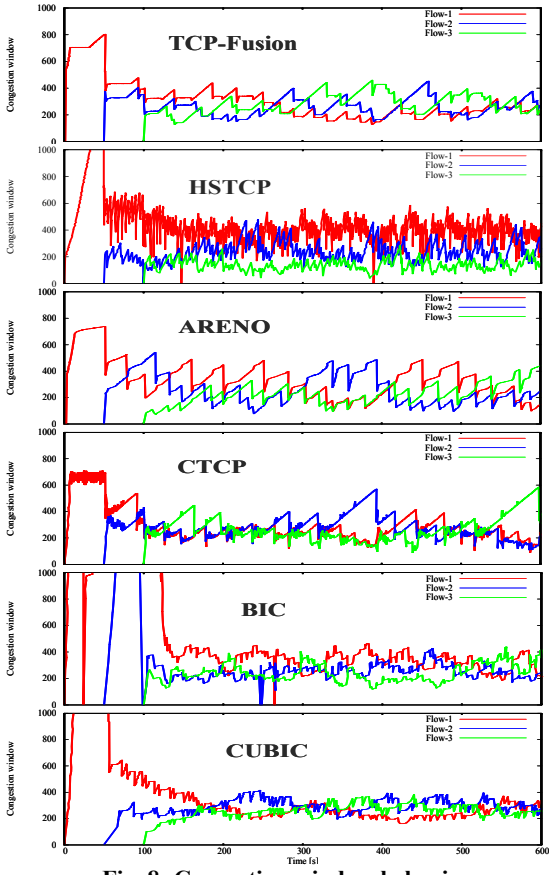

Fig. 8: Congestion window behavior; three identical flows are competing in Route 1

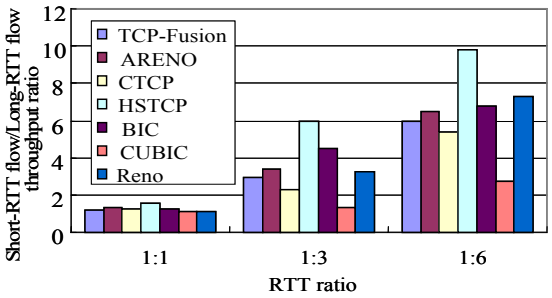

Fig. 9: Throughput ratio of two flows with different RTT in Route 1

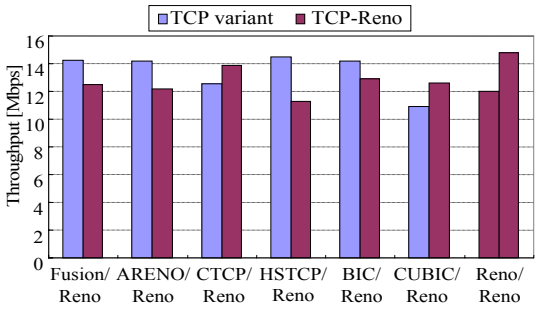

Fig. 10: Throughput when TCP variant and TCP-Reno are competing in Route 2

28.69Mbps and 16.54Mbps, respectively. Likewise, in TCP-Fusion, their throughputs are 36.62Mbps, 31.15Mbps and 33.45Mbps, respectively.

2) Fairness with different RTT flows in Route 1

In this experiment, two identical TCP variant flows are competing with different RTT by using the network emulator. One is short-RTT flow with 20ms RTT and the other is long-RTT flow with 20ms, 60ms or 120ms RTT. Fig. 9 shows the throughput ratio of the short-RTT flow to the long-RTT flow. Although the longer RTT flow gets smaller bandwidth as TCP nature in all TCP variants, especially HSTCP suffers from unfairness. TCP-Fusion, ARENO and CTCP have almost the same ratio as TCP-Reno. This is because their architectures are designed to cause the congestion loss by TCP-Reno like window control to ensure friendliness to TCP-Reno. For a cyclic behavior of TCP, since its window size just before a congestion loss is one of the determining factors of its flow throughput, their RTT fairness are almost same as TCP-Reno. On the other hand, CUBIC performs most fairly. This is the specific property of CUBIC whose window control uses elapsed time since the last congestion event.

3) Friendliness in Route 2

For the last actual network experiment, we test the friendliness when TCP-Reno and TCP variants are competing in *Route* 2 with small BDP. Its BDP is estimated to around 50 packets. Note that we could confirm that the single flow of all protocols can achieve the link bandwidth, as is not shown in this paper. Fig. 10 shows the each throughput of all cases. All TCP variants perform in the same manner, namely share fairly with TCP-Reno. In such a low-speed network or small BDP network, TCP variants enter their TCP-Reno mode triggered by the congestion window size, the period between two consecutive loss events or the congestion level estimation based on RTT.

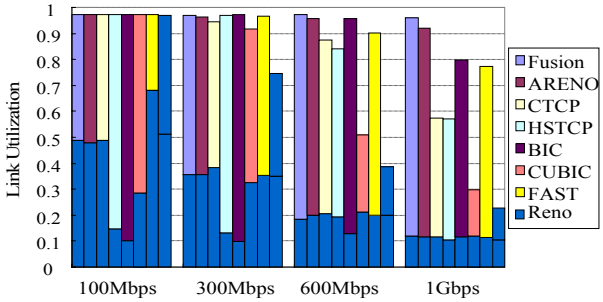

Fig. 11: Throughput when competing TCP variant and TCP-Reno for different link bandwidth (10^{-5} loss rate)

C. Simulation Experiments

In this subsection, we introduce some simulation results that cannot be evaluated in our current implementation experiments; high-speed networks more than 100Mbps and multi-bottleneck links with/without reverse traffic. In simulation experiments, we added FAST to TCP variants team of implementation experiments. All results are obtained using ns-2 simulator [18].

1) Efficiency and Friendliness in high-speed networks

We evaluate the efficiency and friendliness when TCP-Reno and TCP variants flows are competing in high-speed network. The network model is a simple dumb-bell topology such as Fig. 3. The bandwidth and the delay of access links are 1Gbps and 1ms, respectively. The bandwidth and the delay of bottleneck link are 100Mbps, 300Mbps, 600Mbps or 1Gbps and 20ms, respectively. The random packet loss rate at bottleneck link is set to 10^{-5}. The router buffer capacity is set to BDP of each experiment and the router buffers employ *Taildrop* discipline.

Fig. 11 shows the utilization ratio of bandwidth share of each flow. In this figure, the throughput in the case with competing two TCP-Reno flows is also shown at rightmost in each case. The goal in this experiment is coordination with TCP-Reno as well as link utilization improvement. TCP variant flow should not get bandwidth in return of the throughput reduction of the coexisting TCP-Reno flow. TCP-Fusion, ARENO and CTCP perform friendly even in high-speed networks. The utilization ratio of the coexisting TCP-Reno flow is almost the same ratio as those of two TCP-Reno flows competing case. Among of them, TCP-Fusion can also achieve the highest utilization ratio. HSTCP and BIC get more bandwidth by stunting TCP-Reno flow. By contrast, FAST cannot get fair share at 100Mbps, which is too friendly.

2) Fairness in complex links

For the last experiments in this paper, we evaluate the fairness in multi-bottleneck links. The network model is the parking lot topology as shown in Fig. 12. The bandwidth and delay of each link are illustrated in this figure. The router buffer capacity is set to BDP of the longest RTT flow, and the router buffers employ *Taildrop* discipline. In this experiment, there are a total of six flows, three TCP variants and three TCP-Reno, with three different hop counts; S1-S4, S2-S4 and S3-S4, in forward direction. There are also each four TCP-Reno flows on the ACK return path; S5-S6, S6-S7 and S7-S8, respectively, to

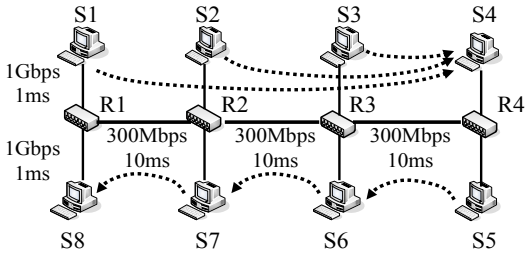

Fig. 12: Multi-bottleneck links topology

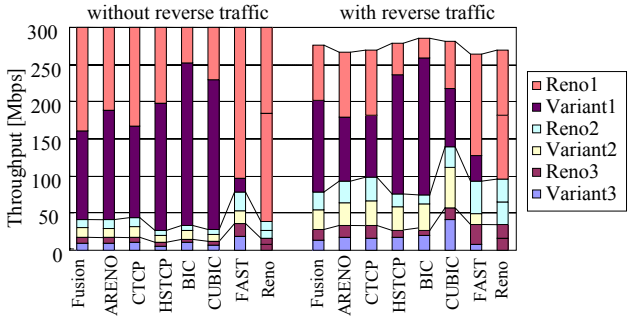

Fig. 13: Throughput of each flow without/with reverse traffic

verify the effect of reverse traffic.

Fig. 13 shows the throughput of each flow without/with reverse traffic at left and right side. Reno1 (2 and 3) and Variant1 (2 and 3) represent TCP-Reno and TCP variant flows with one (two and three) hop counts, respectively. The total throughput means the link utilization of R3-R4, which is the shared link of all forward path flows. Further, to visualize the fairness among different hop counts or different RTT flows, their throughputs are divided into each hop counts flows in this figure.

Whereas all protocols fully utilize the bandwidth in the case without reverse traffic, those of the case with reverse traffic are underutilized. This is because TCP flows experience the number of timeout or retransmission in the presence of the reverse traffic [15]. TCP-Fusion, ARENO and CTCP can also perform friendly to TCP-Reno even in multi-bottleneck links with/without reverse traffic.

We then point out another observation that is the fairness among different hop counts flows or different RTT flows with/without reverse traffic. While their fairness among different RTT flows are not very enough in the absence of reverse traffic, those of the cases in the presence of the reverse traffic are comparatively favorable. Especially the flows with two hop counts get larger throughputs in return for reduction of one hop counts flow throughputs. We suppose that the flows having the most bandwidth share come under the more influence of the cross traffic such as reverse traffic, which results in improving the fairness. We will carry out further investigation of this problem as future works.

IV. Conclusion

In this paper, we present a new hybrid congestion control algorithm, called TCP-Fusion. This protocol integrates three characteristics; TCP-Reno, TCP-Vegas and TCP-Westwood, which provides good balance between efficiency and friendliness even in high-speed networks with large bandwidth-delay product and non-negligible random packet losses. Our implementation and simulation results show that TCP-Fusion can achieve the highest throughput in existing protocols. Moreover, when a TCP-Fusion flow competes with a TCP-Reno flow, it can obtain more than fair share when there is unused residual capacity otherwise, it shares the same bandwidth to coexisting flows. We also emphasize that the fairness among TCP-Fusion flows is almost same as that of TCP-Reno.

As future work, our first job is more investigation to optimally choose a parameter such as D_{min}. Moreover, we are planning more extensive implementation experiments over high-speed networks with multi-bottleneck links and more realistic network environment with short-lived flows and reverse traffic.

Acknowledgement

The authors would like to thank Prof. Goto and Prof. Kameyama, Waseda University, Japan, for their help in setting up the actual network experiments.

References

[1] W. Richard Stevens: "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms," IETF RFC 2581, 1997.
[2] Janey C, Hoe: "Improving the Start-up Behavior of a Congestion Control Scheme for TCP," In Proc. of ACM SIGCOMM 1996, August 1996.
[3] S.Floyd: "Highspeed TCP for Large Congestion Window", IETF RFC3649, 2003.
[4] Tom Kelly: "Scalable TCP: Improving Performance in High-speed Wide Area Networks." In PFLDnet 2003.
[5] C. Casetti, M. Gerla, S. Mascolo, M. Y. Sanadidi, and R. Wang: "TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links", In proc. of ACM Mobicom 2001.
[6] R. Wang, M. Valla, M.Y. Sanadidi, B. K. F. Ng, and M. Gerla: "Efficiency/Friendliness Tradeoffs in TCP Westwood", Seventh IEEE Symposium on Computers and Communications, 2002.
[7] L.S. Brakmo and L.L. Perterson: "TCP Vegas: End-to-End Congestion Avoidance on a Global Internet," IEEE Journal on Selected Areas in Communication, Vol. 13, Nov. 8, 1995.
[8] Cheng Jin, David X. Wei and Steven H. Low: "FAST TCP: motivation, architecture, algorithms, performance", In proc. of INFOCOM 2004.
[9] H. Shimonishi, T. Hama and T. Murase: "TCP-Adaptive Reno for Improving Efficiency-Friendliness Tradeoffs of TCP Congestion Control Algorithm", In proc. of PFLDnet 2006.
[10] H. Shimonishi, M. Y. Sanadidi, and M. Gerla: "Improving Efficiency-Friendliness Tradeoffs of TCP in Wired-Wireless Combined Networks", In proc. of ICC, 2005.
[11] K. Tokuda, G. Hasegawa and M. Murata: "Performance analysis of HighSpeed TCP and its improvement for high throughput and fairness against TCP Reno connections", in Proc. of HSN 2003.
[12] R. King, R. Baraniuk and R. Riedi: "TCP-Africa: An Adaptive and Fair Rapid Increase Rule for Scalable TCP", In Proc. of INFOCOM 2005.
[13] K. Tan. J. Song, Q. Zhang, and M. Sridharan: "Compound TCP: A Scalable and TCP-Friendly Congestion Control for High-speed Networks", in Proc of PFLDnet 2006.
[14] L. Xu, K. Harfoush and I. Rhee: "Binary Increase Congestion Control for Fast, Long Distance Networks", in Proc of INFOCOM 2004.
[15] S. Mascolo and F. Vacirca: "The effect of reverse traffic on the performance of new TCP congestion control algorithms", in Proc of PFLDnet 2006.
[16] I. Rhee and L. Xu: "CUBIC: A New TCP-Friendly High-speed TCP Variant", in Proc of PFLDnet 2005.
[17] K. Kaneko and J. Katto: "Reno Friendly TCP Westwood based on Router Buffer Estimation", in Proc of ICAS/ICNS 2005.
[18] "ns-2 network simulator(ver.2)," http://www.mash.cs.berkley.edu/ns.
[19] "Iperf", http://dast.nlanr.net/Projects/Iperf/