

Development of a High-speed Transport Protocol with TCP-Reno Friendliness

Junichi MASAKI, Nishantha G.G.D.*, Yukuo Hayashida

Saga University, *Ritsumeikan Asia Pacific University

Saga-Japan 840-1050, Oita-Japan 874-8577

hayasida@is.saga-u.ac.jp, *gamagelk@apu.ac.jp

Abstract—The traditional TCP-Reno is not capable of achieving a high throughput in high-speed networks due to its *highly conservative and loss-dependent* congestion control mechanism. Although TCP-Vegas achieves a good link utilization due to its proactive type of congestion control mechanism that uses RTT as an indication of network congestion, it lacks friendliness to TCP-Reno. Protocols such as HighSpeed-TCP and Scalable-TCP have been proposed to address this problem, but they behave non-friendlily to TCP-Reno especially when sharing high bandwidth links. In order to improve the friendliness with TCP-Reno and throughput performance in high-speed networks, we propose a new congestion control mechanism, *TCP-xHS (High-speed)* that harmonizes the fast ramping and proactive congestion controlling approaches of HighSpeed-TCP and TCP-Vegas. Through simulation results we justify the effectiveness of TCP-xHS for a wider range of link bandwidths.

Keywords - TCP-Reno, Highspeed-TCP, TCP-Vegas, Fairness

I. INTRODUCTION

Rapid proliferation of broadband networks with the emergence of technologies such as optical fiber and high-speed ADSL (Asymmetric Digital Subscriber Line) give rise to an array of applications demanding high bandwidth (e.g., large file-transfer, delivery of high-quality video streaming etc.) It is well known that the traditional reliable transfer protocol, TCP-Reno [1], cannot fully utilize the bandwidth of high-speed networks due to its highly conservative and reactive congestion control mechanism to packet loss. To improve the throughput performance, HighSpeed-TCP [2] and Scalable-TCP [3] have been proposed to use more aggressively control than that used in TCP-Reno. But, this aggressiveness results in degrading the throughput of TCP-Reno in a shared link. In a different approach, to achieve higher throughput than that of TCP-Reno in high bandwidth situations, TCP-Vegas [4] uses a mechanism that has an ability of detecting network congestion by measuring the round-trip time (RTT) as an indication of network congestion. But, the performance of TCP-Vegas is suppressed by the greedy nature of TCP-Reno when the bottleneck link is shared with each other, resulting unfairness to TCP-Vegas.

In this paper, we first investigate the properties of the congestion control mechanisms of TCP-Reno, TCP-Vegas, HighSpeed-TCP, and Scalable-TCP which are the key

protocols for deriving the novel algorithm. Next, paying attention to the aggressive increasing (*fast ramp-up*) of HighSpeed-TCP and the mechanism of RTT-Based network congestion detection in TCP-Vegas, we propose a new congestion control mechanism, *TCP-xHS (High-speed)*, that achieves a high throughput in high-speed networks and ensures friendliness to TCP-Reno for a wider range of link bandwidths. The simulation results show that the proposed congestion control mechanism performs better than the high speed TCP versions mentioned above.

The novel algorithm we propose in this paper harmonizes different operating regions of TCP-Reno, HighSpeed-TCP and TCP-Vegas to form one effective algorithm, and hence the name TCP-xHS. We stick to HighSpeed-TCP style of fast ramping at low loss rates to catch up the bandwidth quickly while implementing TCP-Vegas style of RTT-based congestion awareness to enforce brakes against the aggressively increase rate. This slows down the fast ramping at rate increase allowing other competing flows to increase sending rates without experiencing packet loss. Moreover, TCP-xHS stays friendly to a competing TCP-Reno connection while it grasps any extra bandwidth left unused by TCP-Reno.

Section II of this paper presents a detail overview of Traditional TCPs, TCP-Reno and TCP-Vegas. Section III extends this discussion to existing high speed protocols. The novel algorithm, TCP-xHS, is explained in Section IV. Finally Section V discusses the simulation results followed by conclusions.

II. TRADITIONAL TCPs

A. TCP-Reno

The congestion control mechanism of TCP-Reno is governed by two phases: Slow-Start (SS) phase and Congestion-Avoidance (CA) phase. In SS phase, the current window size (*cwnd*) is increased by one packet for each receiving ACK, resulting exponential increasing. When *cwnd* is over the slow-start threshold (*ssthresh*), the phase transits from SS to CA, changing the exponential increase into a linear increase. that is:

$$cwnd = cwnd + 1 \text{ in SS} \quad (1)$$

$$cwnd = cwnd + 1/cwnd \text{ in CA} \quad (2)$$

TCP-Reno uses packet loss events as an indication of congestion, leading to two congestion states: time-out and triple duplicate ACKs as explained in equations 3 to 5. On receiving the 3Dup ACKs, $cwnd$ is reduced to half of its value, whereas in timeout state the value of $cwnd$ is reduced to 1. The value of $ssthresh$ is set to half of $cwnd$ both in 3Dup ACK and timeout states.

$$ssthresh = cwnd/2 \text{ for timeout or triple ACK, } (3)$$

$$cwnd = cwnd/2 \text{ for triple ACK, } (4)$$

$$cwnd = 1 \text{ for timeout. } (5)$$

1) *Sack option*: Suppose that multiple packets are lost in one RTT. As TCP-Reno is using the sliding window mechanism based on the cumulative acknowledgements, the first packet lost in one round can be detected by a triple ACK. Therefore, detection of immediately subsequent packet losses can be done only when the ACKs for re-transmitted packets are received. That is, TCP-Reno can only detect one lost packet in one RTT. To detect burst packet losses, the selective ACK method was considered using an optional TCP header field, called *Sack option*. In all our arguments below, we assume TCP-Reno with sack-option, because this option is adopted by default in most of the current TCP implementations.

B. TCP-Vegas

TCP-Vegas also consists of two operating phases: SS and CA, but uses a different mechanism compared to TCP-Reno to sense network congestion and to control $cwnd$. TCP-Vegas does not use the packet loss event, but the measured-RTT as an indication of congestion. TCP-Vegas treats an increase of measured RTT as initial stage of network congestion. That is, RTT is increased as TCP connections start using the buffers of the bottleneck link router as the packets flow in the network rise beyond the capacity.

TCP-Vegas estimates expected and actual throughput of a connection based on two RTT estimations as given in the following equations. Here, $BaseRTT$ is defined as RTT under no congestion, which is usually set to the minimum value of the measured RTT .

$$Expected = WindowSize/BaseRTT (6)$$

$$Actual = WindowSize/RTT (7)$$

From the above two equations, the queue size of the bottleneck router for the given TCP connection is calculated as follows:

$$Diff = (Expected - Actual) * baseRTT (8)$$

The window control mechanism of TCP-Vegas is performed with parameters α , β , and $Diff$ in every RTT as given below:

- $if(Diff < \alpha) cwnd = cwnd + 1$: i.e., $cwnd$ is increased based on the inference of lightly-loaded channel.

- $if(\alpha < Diff < \beta) cwnd = cwnd$: i.e., $cwnd$ is not changed based on the inference of moderately-loaded channel.
- $if(\beta < Diff) cwnd = cwnd - 1$: i.e., $cwnd$ is decreased based on the inference of heavily-loaded channel.

The two thresholds α and β are set so as to control the queue size of the bottleneck router effectively. Having a thin margin for the allowed buffers in the backlog, i.e. from α to β , TCP-Vegas can achieve high stability. Higher throughput can also be achieved due to proactive congestion controlling that prevents packet losses as much as possible.

In the phase of SS, $cwnd$ is increased on ACK reception. The value of increment is half of the increment of TCP-Reno. When $Diff > \gamma$, the phase transits from SS to CA. ($\alpha = 1$, $\beta = 3$, and $\gamma = 1$ are the typical values for TCP-Vegas).

When ACK is received in phase SS

$$cwnd = cwnd + 1/2 (9)$$

$$if(Diff > \gamma) \text{ then SS to CA. } (10)$$

1) *Issue of TCP-Vegas*: TCP-Vegas achieves high stability and high throughput performances, but receives a smaller share when the bottleneck link is shared with TCP-Reno. This is because TCP-Reno cannot decrease $cwnd$ without packet loss, while TCP-Vegas has to decrease $cwnd$ to prevent packet loss. Therefore, TCP-Reno connections may rob TCP-Vegas available bandwidth. We cannot ignore this problem because most of TCP connections in practice are TCP-Reno.

III. HIGH-SPEED TRANSPORT PROTOCOLS

TCP-Reno cannot achieve high throughput in high-speed networks. Since the window increase in TCP-Reno is too smaller than the window decrease on packet loss detection, TCP-Reno style of window ramping can hardly increase the window size to very higher levels. Following these observations, two high-speed TCP algorithms are proposed in the literature, HighSpeed-TCP [2] and Scalable-TCP [3].

A. HighSpeed-TCP

When $cwnd < W_{low}$, the window control algorithm of HighSpeed-TCP follows the same $cwnd$ locus of TCP-Reno. That is the additive increase and multiplicative decrease parameters are the same and are constants. However, when $cwnd > W_{low}$, the algorithm starts varying the above two parameters as a function of its window size to scale the parameters to suit the high speed rates. The behavior of the window dependent increasing and decreasing components with window size is as illustrated in figure 1. Accordingly, the increase and decrease amounts change dynamically with the window size as explained by the following equations. The congestion window behavior in HighSpeed-TCP is illustrated in figure 2.

$if(cwnd < ssthresh)$

$$cwnd = cwnd + 1, \text{ in SS } (11)$$

else
 $cwnd = cwnd + A$, in CA

When the loss-event occurs,
 if (tripleACK)

$$ssthresh = cwnd * (1 - B), \quad (12)$$

$$cwnd = cwnd * (1 - B) \quad (13)$$

else if (timeout)

$$ssthresh = cwnd/2, \quad (14)$$

$$cwnd = 1 \quad (15)$$

given that,

if ($cwnd > W_{low}$)

$$A = a(cwnd)/cwnd,$$

$$B = b(cwnd) \quad (16)$$

else *RenoAlgorithm* is applied.

where $a(w)$, $b(w)$, and $p(w)$ are given by:

$$a(w) = \frac{2w^2 * b(w) * p(w)}{2 - b(w)}$$

$$b(w) = \frac{\log w - \log W_{low}}{\log W_{high} - \log W_{low}} (b_{high} - 0.5) + 0.5 \quad (17)$$

$$p(w) = \exp \left[\frac{\log w - \log W_{low}}{\log W_{high} - \log W_{low}} (\log P_{high} - \log P_{low}) + \log P_{low} \right]$$

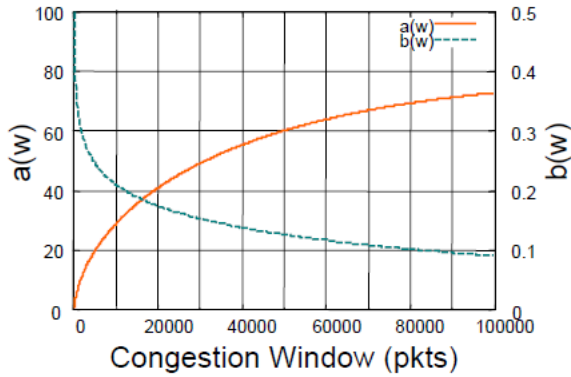


Fig. 1. Congestion Window vs. $a(w)$ and $b(w)$

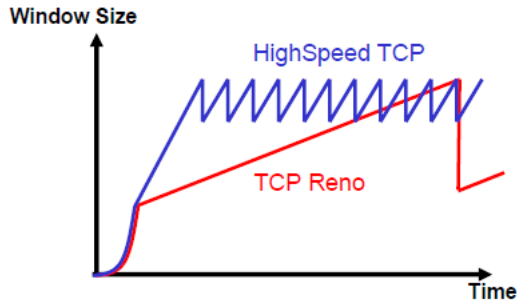


Fig. 2. Congestion Window behavior in HighSpeed-TCP

In eqn. (18), P_{high} , W_{high} , b_{high} and W_{low} are the control parameters of HighSpeed-TCP [2]. Here, $P_{low} = 1.5/W_{low}^2$

is the packet loss rate when the average window size of TCP-Reno becomes W_{low} . The average window size becomes W_{high} when the packet loss rate is P_{high} , and the window size is decreased to $(1 - b_{high})W_{high}$ on detecting tripleACK when the window size is larger than W_{high} .

B. Scalable-TCP

The window control algorithm of Scalable-TCP enforces more aggressive increasing, and less aggressive decreasing compared to the controls in HighSpeed-TCP. The Scalable-TCP algorithm is as follows:

if ($cwnd < ssthresh$)

$$cwnd = cwnd + 1, \text{ in SS} \quad (19)$$

else

$$cwnd = cwnd + A, \text{ in CA} \quad (20)$$

When the loss-event occurs,

if (tripleACK)

$$ssthresh = cwnd * (1 - B), \quad (21)$$

$$cwnd = cwnd * (1 - B) \quad (22)$$

else if (timeout)

$$ssthresh = cwnd/2, \quad (23)$$

$$cwnd = 1 \quad (24)$$

given that,

if ($cwnd > W_{low}$)

$$A = 0.01,$$

$$B = 0.125 \quad (25)$$

else *RenoAlgorithm* is applied.

Unlike TCP-Reno and HighSpeed-TCP, Scalable-TCP does not relate the increase parameter A to $cwnd$, but it uses a constant increase value similar to that used in Slow-Start phase. In other words, Scalable-TCP adopts that the increase of $cwnd$ is not linear but exponential in Congestion-Avoidance phase as shown in figure 3.

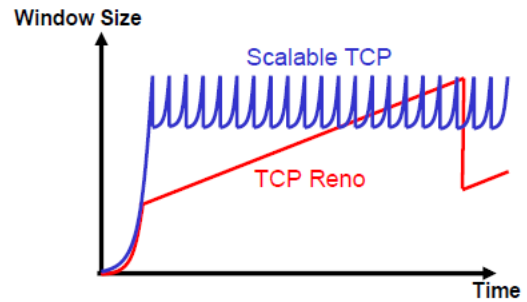


Fig. 3. Congestion Window behavior in Scalable-TCP

C. Issue of high-speed transport protocols

High-speed transport protocols achieve higher utilization rates in high-speed networks following a common methodology: large window increase under non-congestion and small windows decrease under light congestion. However, consequently, the connections of High-speed transport protocols may steal bandwidth from competing TCP-Reno connections due to this asymmetric increase and decrease behavior. Considering the fact that majority of connections are TCP-Reno in the real networks, current communication setup will be largely effected by high-speed transport protocols (i.e. throughput degradation of TCP-Reno). Hence, development of TCP which is adaptable for a high-speed networks and friendly to TCP-Reno is a demanding and highly focused research area.

IV. TCP-xHS

In this section, we propose the novel algorithm TCP-xHS that can achieve a high throughput while delivering fairness to TCP-Reno in a wider range of link bandwidths. TCP-xHS is designed by carefully observing the behaviors of three known reference algorithms: TCP-Reno, HighSpeed-TCP and TCP-Vegas. It is designed to operate differently at different levels of network congestion by closely mimicking the behaviors of the reference protocols. This makes our design objective driven and increases the visibility of the algorithmic behavior of TCP-xHS as the behaviors of the reference protocols are already well known.

Just like TCP-Vegas, TCP-xHS adopts RTT as an indication of the congestion level. When RTT is small, the window size is aggressively increased adopting the behavior of HighSpeed-TCP because the link is not used enough. When RTT is large, the window size is controlled by adopting the behavior of TCP-Vegas in order to keep the network stable. Additionally, TCP-xHS estimates its window size by comparing two auxiliary congestion windows: (a) high-speed window ($cwnd_{HS}$) using $Diff$ following TCP-Vegas algorithm and (b) TCP-Reno window ($cwnd_{reno}$) following TCP-Reno algorithm. In order to ensure that the TCP-xHS stays friendly to TCP-Reno and occupies any vacant bandwidth that TCP-Reno can not utilize, the congestion window is decided by comparing $cwnd_{HS}$ with $cwnd_{reno}$ and taking the higher value as $cwnd$. TCP-xHS algorithm is explained in detail below.

In *SS*, TCP-Reno's exponential increasing method is used. For each ACK received:

$$cwnd = cwnd + 1 \quad (26)$$

$$cwnd_{reno} = cwnd_{HS} = cwnd \quad (27)$$

$$if(cwnd > ssthresh) \text{ then } SS \text{ to } CA. \quad (28)$$

In *CA*, for each ACK received:

$$cwnd_{reno} = cwnd_{reno} + 1/cwnd \quad (29)$$

$$cwnd_{HS} = cwnd_{HS} + A \quad (30)$$

Eqn.(29) and eqn.(30) follow the default behaviors of TCP-Reno and HighSpeed-TCP respectively, under no congestion. On loss detection, the two auxiliary window sizes are estimated according to the behavior of the respective algorithms as given below.

When the loss-event occurs,

$$ssthresh = cwnd/2 \quad (31)$$

if (tripleACK)

$$cwnd_{reno} = cwnd/2$$

$$cwnd_{HS} = cwnd * (1 - B) \quad (32)$$

else if (timeout)

$$cwnd_{reno} = cwnd_{HS} = 1 \quad (33)$$

The estimation of $cwnd_{HS}$, is enforced to deviate from the HighSpeed-TCP locus after detecting a certain level of congestion. Here the number of buffers in the backlog, estimated as in TCP-Vegas, is used to judge whether the link is congested or not. That is:

B is set to eqn.(17), and A is depend on ω , i.e.,

if ($Diff < \omega$) A is set to eqn.(17),

else $A = -(Diff - \omega)/cwnd$.

where $Diff$ is the extra buffers in the bottleneck router as calculated in TCP-Vegas and ω is set to 30 based on heuristic observations.

In other words, if $Diff < \omega$, the window size is aggressively increased like in HighSpeed-TCP because the link is not used fully. When $Diff > \omega$, the window size is decreased because the link is thought to be overused. As a result, TCP-xHS waits until $cwnd_{reno}$ catches up with $cwnd_{HS}$ by keeping $Diff$ (the queue size of the bottleneck router) below ω buffers. After TCP-Reno catches up its fare share, TCP-xHS increases its window size at the same pace with TCP-Reno. This behaviour is illustrated in figure 4. Here, the control parameters $P_{high}, W_{high}, b_{high}$ and W_{low} are used as in HighSpeed-TCP.

Finally, the maximum of the two window sizes is selected as the congestion window, $cwnd$, for TCP-xHS.

$$cwnd = \max(cwnd_{HS}, cwnd_{reno}) \quad (34)$$

V. PERFORMANCE EVALUATION

In this section, we evaluate TCP-xHS performance using the Network-Simulator2 (NS-2) [9] and compare the performance with other high speed TCP versions. We limit out evaluation to two main criteria, link utilization and friendliness to TCP-Reno.

$$LinkUtilizationRate(\%) = \frac{AverageThroughput}{BottleneckBandwidth} * 100 \quad (35)$$

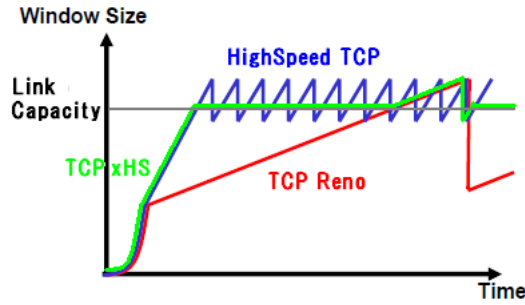


Fig. 4. Congestion window behavior in TCP-xHS

The simulations were conducted with following protocol settings. Except for TCP-Vegas, all TCP versions were operating with the Sack-option enabled. For TCP-Reno, standard parameters were used. TCP-Vegas was used with its typical values $\alpha = 1$, $\beta = 3$, and $\gamma = 1$. HighSpeed-TCP was used with its typical parameters $P_{high} = 1.0 * 10^{-7}$, $W_{high} = 83000$, $b_{high} = 0.1$, and $W_{low} = 38$. For Scalable-TCP, $W_{low} = 100$ was used. For TCP-xHS, the value of ω was set to 30 buffers based on heuristic observations.

A. Throughput performance of a single connection

Simulations with a single connection evaluate the independent performance of each TCP version for the given bottleneck link. Figure 5 shows the network model used in this scenario. Each connection is observed for 1000 seconds from the start. Parameters such as the transmission delay, the buffer size of the bottleneck router and the bandwidths of the connecting links are fixed as indicated. Simulation is conducted for four different bottleneck bandwidths (i.e. 1Mbps, 10Mbps, 100Mbps and 1000Mbps.)

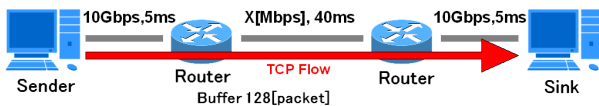


Fig. 5. Network model for a single TCP connection

Figure 6 illustrates the link utilization for each TCP version under the discussion. Each TCP can raise the rate up to the link capacity of 10Mbps. The problem of TCP-Reno in catching up high bandwidth is visible beyond 10Mbps. TCP's rate halving and timeout based loss recovery becomes a dominant negative effect as the link capacity increases. TCP-Vegas also suffers from the same problem at higher bandwidths (above 100Mbps). Although TCP-Vegas's conservative behavior against packet loss allows it to go for higher sending rates than TCP-Reno, at higher bandwidths, TCP-Vegas also experiences a slower recovery from lower window size like TCP-Reno. Both Scalable-TCP and HighSpeed-TCP still keep at higher rates due to fast ramp up. The proposed algorithm TCP-xHS shows the best link utilization.

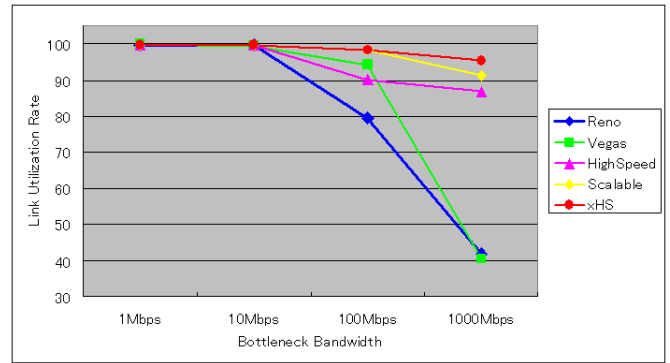


Fig. 6. Link-Utilization TCP versions for varying link capacity

Congestion window evolution of each TCP version is illustrated in figure 7 for bottleneck link capacity of 100Mbps. Compared to TCP-Reno, it can be observed that the window size increase is more aggressive and the window size decrease is less aggressive in both HighSpeed-TCP and Scalable-TCP algorithms. This behavior results in shorter bursts of packet loss at a higher frequency. Due to TCP-Vegas RTT based controlling, the window size is kept unchanged as the operating region for a single connection falls mostly inside the stable mode. TCP-xHS increases aggressively like HighSpeed-TCP when the link is not fully utilized, and it keeps the window size stable as in TCP-Vegas when the link is utilized with allowable buffers in the backlog. Moreover, TCP-xHS increases linearly like TCP-Reno after TCP-Reno catch up the fair-share. Therefore in TCP-xHS, the interval of packet loss is equal to that of TCP-Reno. In addition, the rate decreasing behavior on packet loss is as same as that of HighSpeed-TCP.

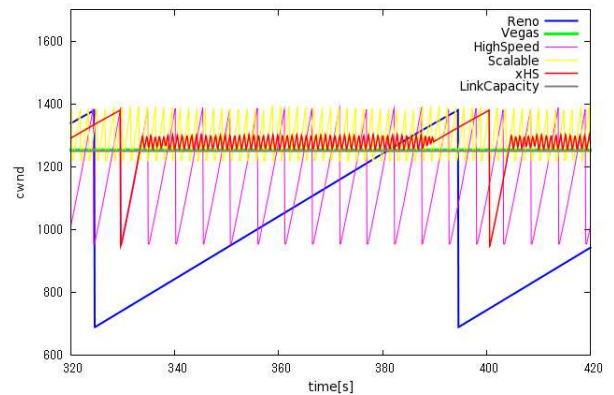


Fig. 7. Congestion window evolution of TCP versions (at 100Mbps)

B. Friendliness with TCP-Reno

In the previous section we showed the link utilization performance of TCP-xHS for a range of link bandwidths. In this section, simulation with the two competing connections is used to evaluate the friendliness of each competing protocol with TCP-Reno. The notation *flow1* is used to denote the competing high speed TCP variant whereas *flow2* is used to denote the competing TCP-Reno connection. The bottleneck

link is shared between *flow1* and *flow2*. Figure 8 illustrates the network model used in the simulation setup. The typical dumbbell shaped network model is used because of its simplicity to model a single bottleneck link without loss of generality. The bottleneck link is varied with bandwidth values from 1Mbps, 10Mbps, 100Mbps and 1000Mbps. Each connection is active for 1000 seconds. The other parameters such as the transmission delays of each link, the buffer size of the bottleneck router and the link bandwidths are as indicated in figure 8.

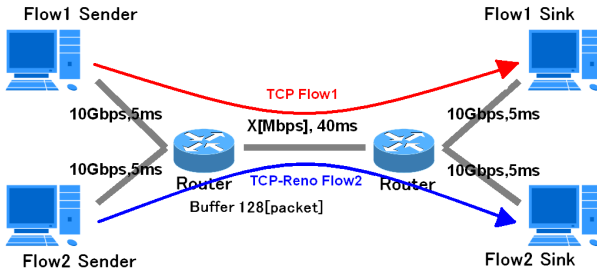


Fig. 8. Network model

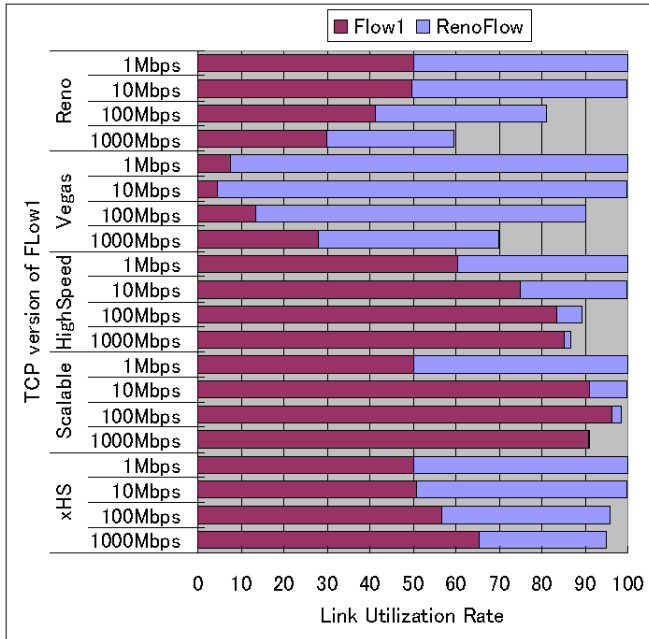


Fig. 9. High-speed TCP variants competing with TCP-Reno

Figure 9 illustrates the throughput performance of competing algorithms in the shared link. At 1Mbps TCP-Reno degrades TCP-Vegas. The others protocols at this low bandwidth are friendly with TCP-Reno. The reason for this friendliness is due to the fact that the permissible outstanding packets per connection flowing through the link is smaller than W_{low} . Hence, the window control algorithms of HighSpeed-TCP and Scalable-TCP performs equally to TCP-Reno.

As the bottleneck bandwidth is increased towards 1Gbps, it can be seen that both HighSpeed-TCP and Scalable-TCP

render an increasing unfairness to TCP-Reno. This is partly due to the high aggressive behavior of the competing protocols and partly due to TCP-Reno's low link utilization. It can be also observed that TCP-Vegas starts capturing a good share in the link since TCP-Vegas less aggressive decreasing behavior becomes dominant at higher bandwidths. TCP-xHS remains gentle to TCP-Reno in a wider bandwidth range. TCP-xHS always behave friendly to TCP-Reno and use the surplus band that TCP-Reno cannot use. At 1Gbps, TCP-xHS achieves high throughput by means of using this surplus band that results in a link utilization rate of 95%. Although HighSpeed-TCP and Scalable-TCP achieve high throughput, friendliness with TCP-Reno turns worse as the bottleneck becomes thicker.

VI. CONCLUSION

In this paper, we proposed TCP-xHS which performs superior to existing high-speed TCP protocols and through NS-2 based simulations we evaluated the performance in terms of link utilization and TCP-friendliness. The simulation results show that its congestion control mechanism is more effective than the traditional high-speed transport protocols. However, the simulations conducted over a limited scope in this work is necessary to be extended to more complex network models. For example, a network model encompassing wireless links and dynamic routing will be subject to study in future research.

REFERENCES

- [1] Jitendra Padhye, Victor Firoiu, Donald F. Towsley, and James F. Kurose: "Modeling TCP Reno performance: a simple model and its empirical validation", IEEE/ACM Transactions on Networking (TON), Volume 8, Issue 2, pp. 133-145, April 2000. http://www.cs.iastate.edu/~cs610/Modeling_TCP_Throughput_Umass.pdf
- [2] Floyd, S., Ratnasamy, S. and Shenker, S., "Modifying TCP's congestion control for high speeds," Internet Draft, May 2002 <http://www.icir.org/floyd/papers/hstcp.pdf>
- [3] T. Kelly, "Scalable TCP : Improving performance in highspeed wide area networks," in SIGCOMM, 2003 http://www.deneholme.net/tom/papers/scalable_improve_hswan.pdf
- [4] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "TCP Vegas: New techniques for congestion detection and avoidance," in SIGCOMM, 1994 <ftp://ftp.cs.arizona.edu/xkernel/Papers/vegas.ps>
- [5] M. TSURU, K. KUMAZOE and Y. OIE, "Towards High Performance TCP for Fast Long-Distance Networks," in IPSJ, 2003 <http://infonet.cse.kyutech.ac.jp/paper/2003/MasatoTsuru-200309-tcp.pdf>
- [6] K. TOKUDA, G. HASEGAWA, and M. MURATA, "Performance Analysis of HighSpeed TCP and its Improvement for High Throughput and Fairness against TCP Reno CConnections," in IEICE, 2003 <http://www.anarg.jp/achievements/web2002/pdf/domestic/kouchit03IN-HighSpeedTCP.pdf>
- [7] T. IGUCHI, G. HASEGAWA, and M. MURATA, "A New Congestion Control Mechanism of TCP with Inline Network Measurement," in IEICE, 2004 <http://www.anarg.jp/achievements/web2004/papers/t-iguti04in-CongestionControlInline.pdf>
- [8] T. IGUCHI, G. HASEGAWA, and M. MURATA, "A New Congestion Control Mechanism of TCP based on Bandwidth Measurement, in IEICE, 2005" <http://www-mura.ist.osaka-u.ac.jp/achievements/web2004/papers/t-iguti05in-CongestionCongtrolMeasurement.pdf>
- [9] "Ns2 network simulator," <http://www.isi.edu/nsnam/ns/>