The Influence of the Large Bandwidth-Delay Product on TCP Reno, NewReno, and SACK

Haewon Lee*, Soo-hyeoung Lee, and Yanghee Choi

School of Computer Science and Engineering

Seoul National University

San 56 - 1, Shilim-dong, Kwanak-ku, Seoul, Korea

TEL: +82 - 2 - 876 - 7170, FAX: +82 - 2 - 876 - 7171

{hwlee,shlee,yhchoi}@mmlab.snu.ac.kr

Abstract

This paper presents results from a series of simulation experiments designed to compare the performance of TCP Reno, NewReno, and Selective Acknowledgement (SACK) on the large bandwidth-delay product (LBD) network. Using the ns-*2 network simulator, we performed simulations with a variety of traffic scenarios using the bandwidth-delay product and the bottleneck buffer size as control parameters.*

The following results are obtained from the simulations. First, increasing the bandwidth-delay product leads to performance degradation regardless of TCP versions and the bottleneck buffer size. Second, NewReno outperforms Reno and SACK when no packet losses occur during the slow-start phase. Finally, increasing the bottleneck buffer size can lead to improve the link utilization, especially as the bandwidth-delay product gets larger.

Keywords: TCP, LBD, Reno, NewReno, SACK

Designated technical subject : Communication Protocols and Architecture

*Please send all correspondence to Haewon Lee

1 Introduction

In order to support the new applications that require large bandwidth and real-time delivery, such as streaming applications or virtual reality, the link capacity of the Internet backbone is growing exponentially. In addition, high-speed links (optical fibers), long and variable delay paths (satellite links), lossy links (wireless network), asymmetric paths (hybrid satellite links, Asymmetric Digital Subscriber Line(ADSL)), and others are widely employed, increasing heterogeneity of the Internet.

In this paper, we focus on the influence of increasing the bandwidth-delay product on the performance of the most popular data transfer protocol in current use, TCP/IP. This is essential not only for network provisioning in the short term (since the rapid growth of Web applications has caused TCP traffic to grow correspondingly) but also for determining how TCP needs to be modified in the longer term.

Several versions of TCP have been proposed. TCP Tahoe is the first version that includes the *congestion avoidance* mechanism [2]. Tahoe adds *fast retransmit* algorithm, which uses the retransmission strategy without waiting for retransmission timeout. Upon receiving three duplicate acknowledgements, Tahoe immediately retransmits unacknowledged segment. TCP Reno [3] evolves from Tahoe and includes the additional algorithm, *fast recovery*, to reach the available bandwidth more quickly than Tahoe after recovering packet losses by *fast retransmit*. TCP NewReno [10] adds the bandwidth-delay product estimation algorithm to Reno for avoiding packet losses during the first slow-start phase. Furthermore, NewReno changes the *fast recovery* algorithm in Reno for the purpose of avoiding unnecessary retransmission timeout when multiple packets are lost in the same window. TCP SACK [4] introduces *Selective Acknowledgement* which reports a non-contiguous set of data that has been received and queued. TCP SACK can recover multiple segment losses in one round-trip time hence achieving better link utilization than Tahoe, Reno, or NewReno. But SACK has a deployment problem because it requires changes in both the sender and receiver protocol suites. TCP Vegas [7] focuses on estimating the bottleneck bandwidth. Vegas always tries to keep the bottleneck buffer occupancy between two thresholds α and β, while Tahoe, Reno, NewReno, or SACK increases the amount of outstanding packets until packet losses occur.

We choose TCP Reno, NewReno, and SACK for performance comparison because Reno has been widely deployed in the Internet and a recent study indicates that 62% of the major web sites in the Internet use TCP NewReno and 41% advertise the "SACK-permitted" option [12].

In most existing data transfer protocols, it is assumed that buffer sizes far exceed the bandwidth-delay product. This assumption may not hold for wide-area networks formed by the interconnection of LANs using high-speed backbone network [6]. Hence we consider the influence of the bottleneck buffer size,

especially when the bottleneck buffer size does not exceed the bandwidth-delay product.

This paper is organized as follows. Section 2 briefly reviews related works. Section 3 describes the simulation environment and performance metrics. Section 4 presents simulation results and analysis. Finally section 5 makes some concluding remarks.

2 Related Work

There has been a lot of researches which consider the performance of TCP in the heterogeneous environment. Barakat [11] gave a brief survey of them.

Jacobson and Braden [1] proposed the *window scale* TCP option in order to efficiently use the available bandwidth on the large bandwidth-delay product networks. With the *window scale* option, TCP window size can be scaled up to 2^{30} bytes. However, expanding the window size in order to match the capacity of an the large bandwidth-delay product may results in a corresponding increase of the probability of more than one packet per window being dropped. This can have a devastating effect upon the throughput of TCP. They addressed this issue and suggested the use of SACK in this environment.

Fall and Floyd [4] evaluated Reno, NewReno, and SACK performance by simulations. However, their works focused on comparing the difference of packet loss recovery mechanisms. In addition, in their simulation, packet loss occurrence was designed specifically to highlight the performance improvement of SACK.

Lakshman and Madhow [6] investigated the performance of TCP with large bandwidth-delay products and random loss. They assumed finite bottleneck buffer size and observed the influence of buffer size change. They performed both analysis and simulations for performance comparison, but they only compared TCP Tahoe with Reno. Furthermore, they only considered the "long-term" behavior of Tahoe and Reno, thus omitted the slow-start behavior of Reno in their model.

Charalambous and Frost [5] performed real experiments for the performance evaluation of TCP Reno, NewReno, and SACK on the high bandwidth-delay network, especially ATM OC-3 and satellite link. They considered both congestion-free performance (in this case packet losses occur due to the bit error) and congested network performance. They also examined the effect of application-level traffic shaping. The limitation of their research was that they focused on the influence of the bit error rate change and the impacts of the bottleneck buffer size was not considered well. In addition, they used only one metric, the throughput, for performance evaluation.

Note that our work differs from the related works in these points:

- We consider the influence of buffer size, especially when the bottleneck buffer size is smaller than the bandwidth-delay product.
- We consider the slow-start effect of Reno, NewReno and SACK.
- We do not assume any specific packet loss model.

3 Simulations

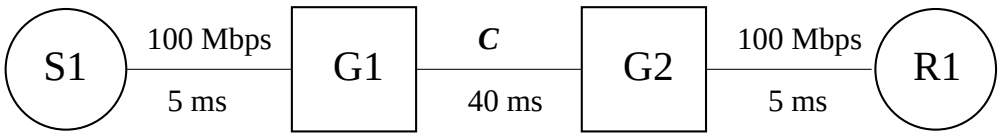


Figure 1: Simulation Topology

We use *ns*-2 [14] for simulation. For all the simulations in this paper, we use a very simple topology shown in Fig. 1 because we model the entire network as the single bottleneck abstraction as our previous work [13]. There are two routers denoted by G1 and G2 and two hosts S1 and R1. The round-trip delay is fixed to 100 millisecond. The link between G1 and G2 is a bottleneck link with capacity denoted as C. In order to change the bandwidth-delay product, We vary C from 1Mbps to 32Mbps.

3.1 Simulation Environment

Following assumptions are made in all the simulations:

- There are no acknowledgement (ACK) losses.
- The TCP receiver does not use delayed acknowledgement.
- The receiver's advertised window size is always greater than the sender window size.
- The sender's initial *ssthresh* value is always greater than the bandwidth-delay product and the slow-start phase is stopped only by packet losses.
- The bottleneck buffer employs FCFS(First-Come-First-Serve) discipline.

We transfer 8 Mbyte from host S1 to R1. The packet size is fixed to 640 bytes. Every simulation starts at simulator clock 0 second.

Table 1: Simulation Parameters

C(Mbps)	$BDP(Kbyte)$	W^**(packet)*	*B(packet)*
1	12.8	20	5, 10, 15, 20
2	25.6	40	10, 20, 30, 40
4	51.2	80	20, 40, 60, 80
8	102.4	160	40, 80, 120, 160
16	204.8	320	80, 160, 240, 320
32	409.6	640	160, 320, 480, 640

The *Bandwidth-Delay Product*(BDP) is defined to be the product of the round-trip delay for a data connection and the capacity of the bottleneck link in its path [1]. The term *Large Bandwidth-Delay Product*(LBD) is somewhat ambiguous and hard to define [1]. High-capacity packet satellite channel is commonly thought as LBD. For example, a DS1-speed satellite channel has a BDP of 10^6 bits or more. Terrestrial fiber-optical paths such as DS3 also has a BDP of 10^6 bits. In this paper we use the term LBD when BDP exceeds 10^6 bits.

We use the term W^* instead of BDP in most cases. W^* is the window size to keep the pipe full,

$$W^* \triangleq \frac{BDP}{packetsize}$$

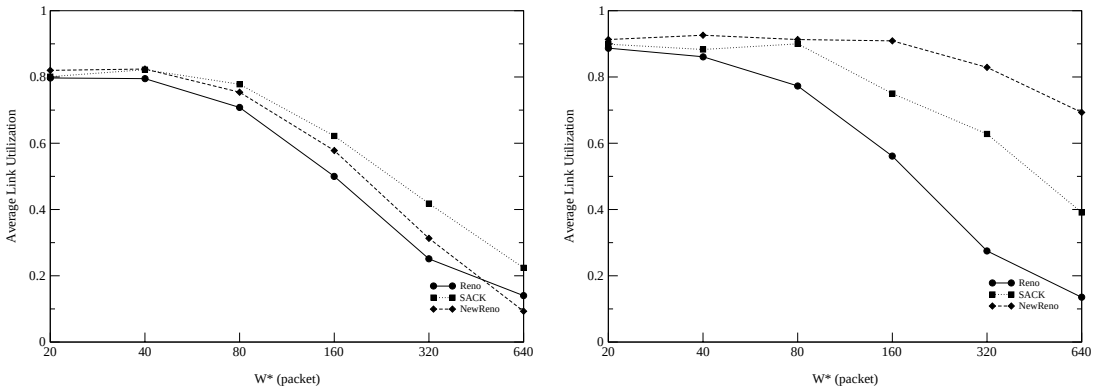
The bottleneck buffer can store up to B packets. We set B to $\frac{1}{4}$, $\frac{1}{2}$, $\frac{3}{4}$ and 1 of W^*. Table 1 shows the simulation parameters in detail.

3.2 Performance Metrics

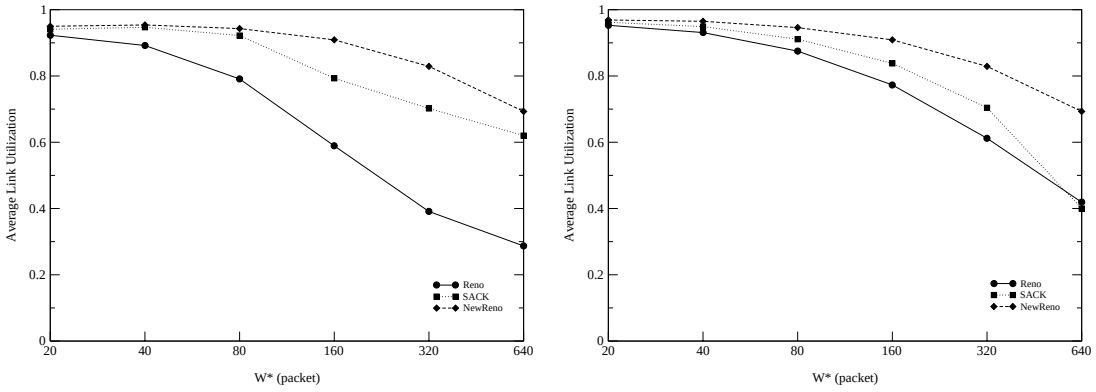
We first define *Transfer Completion Time* (t^c) as the time between the last segment arriving at the receiver and the first segment departing from the sender. $t^c(x)$ is the function of x bits, the time for transferring x bits of data from sender to receiver. Then we can define the *Average Link Utilization* denoted to $\mu(x)$ as the function of x,

$$\mu(x) \triangleq \frac{x}{C \times t^c(x)}$$

$\mu_k(x)$, where $k \in \{Reno, SACK, NewReno\}$ is defined for representing μ of Reno, SACK or NewReno for our convenience. We simply use μ instead of $\mu(64Mbits)$ unless there is confusion with each other because the value of x is fixed to 64 Mbits (8Mbytes * 8) in all the simulations. μ shows how effectively the network is utilized during entire transfer.



(a) $B = \frac{1}{4}W^*$ (b) $B = \frac{1}{2}W^*$



(c) $B = \frac{3}{4}W^*$ (d) $B = W^*$

Figure 2: Average Link Utilization vs. BDP

The *Average Throughput* $T(x)$ is the function of x and defined as similar to μ,

$$T(x) \triangleq \frac{x}{t^c(x)}$$

We also define The *Normalized Average Throughput* $T_N(x)$ in order to compare $T(x)$ with different C as

$$T_N(x) \triangleq \frac{T(x)}{C}$$

For evaluating the influence of increasing B, an additional metric will be defined in the section 4.3.

4 Simulation Results

We first present our main simulation results. We plot μ for investigating the impacts of increasing BDP. Fig. 2 shows that μ has the strong tendency to decrease as BDP increases, regardless of TCP versions

and the size of B. In any simulation settings, μ is the highest when $W^* = 20$ and the smallest when $W^* = 640$.[1]

When we consider the case $B \geqslant \frac{1}{2}W^*$, we see $\mu_{NewReno}$ is always the highest and μ_{Reno} is always the smallest except when $B = W^*$ and $W^* = 640$, in Fig. 2 (d). But we treat them as identical because the difference of μ_{SACK} and μ_{Reno} is just 0.019 and also the difference of t^c of SACK and Reno is only 0.241 seconds.

But when $B = \frac{1}{4}W^*$, not $\mu_{NewReno}$ but μ_{SACK} is the highest. $\mu_{NewReno}$ is higher than μ_{Reno} until $W^* = 320$. But when $W^* = 640$, μ_{Reno} is higher than $\mu_{NewReno}$ and the difference is 0.047. This is not negligible, because t^c of Reno is 14.292 but t^c of NewReno is 21.551 second in this case, so the difference is 7.259 seconds.

We also find that when the buffer size B increases μ also increases by comparing Fig. 2 (a) with (b), (c) and (d). Note that changing B from $\frac{1}{4}W^*$ to $\frac{1}{2}W^*$ causes $\mu_{NewReno}$ to increase significantly, but then $\mu_{NewReno}$ is almost identical despite B increases.

In the following subsections we will discuss below topics in detail:

- The reason why μ decreases as BDP increases.
- The reason why $\mu_{NewReno}$ is smaller than μ_{Reno} when $W^* = 640$ and $B = \frac{1}{4}W^*$.
- The reason why NewReno outperforms Reno and SACK when $B \geqslant \frac{1}{2}W^*$.
- The impacts of increasing the buffer size.

4.1 The Reason Why μ decreases as BDP Increases

In this subsection we explain the reason why μ decreases as BDP increases. We only consider the case when $B = \frac{1}{2}W^*$ because the others are similar except when $B = \frac{1}{4}W^*$ in NewReno. We will explain that in the subsection 4.2. Let's see Fig. 3, 4 and 5.

We first consider T_N of Reno. Note that T_N increases when there are no packet losses and decreases when packet losses occur. Regardless of W^*, T_N increases exponentially in the time interval [0, 1] and decreases abruptly because of the effect of the slow-start and packet losses. T_N decreases until it reaches zero due to retransmission timeout, and exponentially increases again because Reno starts another slow-start.

[1] There are some exceptions as when $W^* = 40$ in Fig. 2 (b), μ_{SACK} is the highest. but we neglect these exceptions because μ, when $W^* = 20, 40$ and 80, are almost identical in these cases.

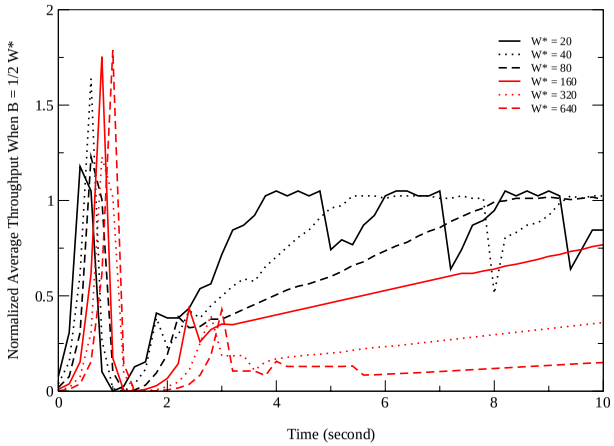


Figure 3: T_N of Reno When $B = \frac{1}{2}W^*$

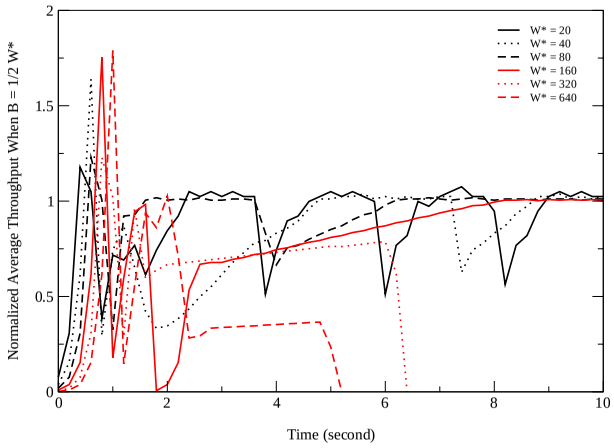


Figure 4: T_N of SACK When $B = \frac{1}{2}W^*$

T_N abruptly decreases again in the time interval [2, 2.4] second because the additional packet loss is detected. In the case when $W^* = 20, 40, 80$ and 160, the packet loss is recovered by *fast retransmit* and no more packet losses occur until T_N exceeds one. When $W^* = 320$, T_N decreases one more time because Reno detects one more packet loss. When $W^* = 640$ Reno suffers an additional packet loss then $W^* = 320$ and enter the congestion avoidance phase at 5.6 second.

Note that when $W^* = 20, 40, 80$ and 160, the time which Reno stops the slow-start phase is roughly identical. But we can see that the slope of T_N is different from one another until T_N reaches one. When $W^* = 20$, the slope of T_N is about 0.4 while the slope is about 0.11 when $W^* = 80$. Table 2 shows that the slope of T_N is roughly in inverse proportion to W^*. It cause μ to decrease as BDP increases because as BDP increases, it takes more time for Reno to reach the bottleneck bandwidth.

In SACK, table 2 shows that the relationship between the slope of T_N and W^* is still held, but there

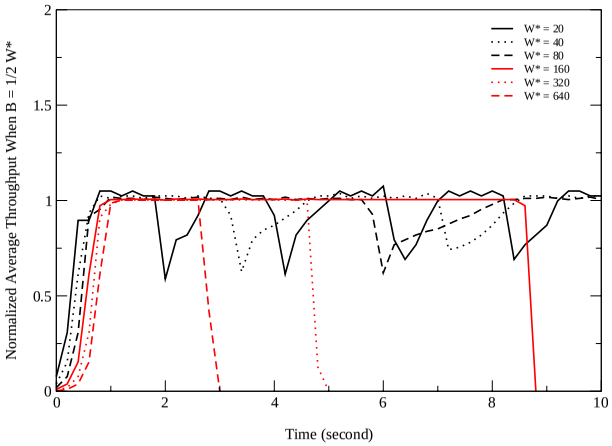


Figure 5: T_N of NewReno When $B = \frac{1}{2}W^*$

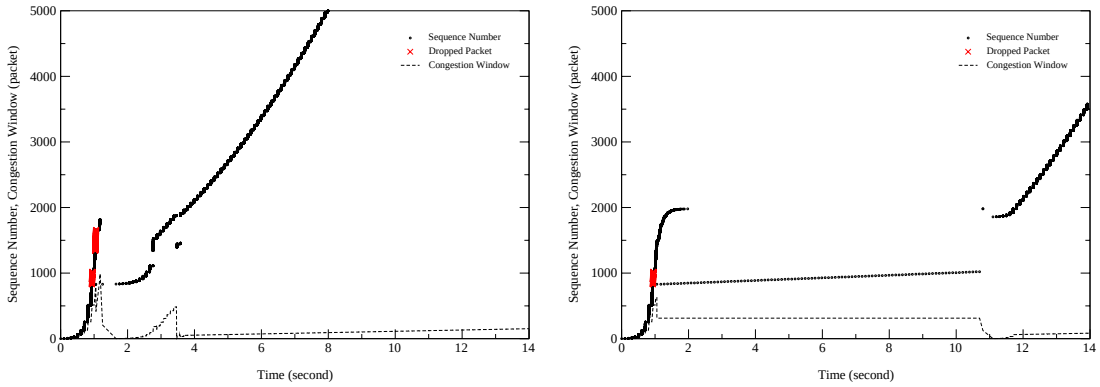
Table 2: The Slope of T_N in Reno and SACK

W^*	$Slope\ (\frac{\Delta T_N}{\Delta t\ (second)})$	
	$Reno$	$SACK$
20	0.422	0.544
40	0.22	0.302
80	0.11	0.149
160	0.064	0.06
320	0.032	0.028
640	0.015	0.015

are several differences from Reno. When we see Fig. 4, T_N when $W^* = 20$ and T_N when $W^* = 80$ is almost identical, because when $W^* = 80$, SACK recovers the packet loss very quickly and reaches the bottleneck bandwidth at 1.8 second while when $W^* = 20$ SACK suffers a packet loss again at 1.6 second so it reach the bottleneck bandwidth at 2.2 second. We can see that in these two cases, μ is almost same in Fig. 2 (b).

When we compare Fig. 4 with Fig. 3, T_N of SACK when $W^* = 40$ is roughly same to T_N of Reno when $W^* = 40$, because Reno and SACK both enter the congestion avoidance phase at 2.0 second. But SACK does not suffer from retransmission timeout, so μ_{SACK} is somewhat higher than μ_{Reno} in this case. We can verify that in Fig. 2 (b).

Fig. 5 shows that in NewReno, T_N increases exponentially and reaches one until 1 second in all the cases. No packet losses occur during the slow-start phase due to its ssthresh estimation algorithm. T_N



(a) Reno (b) NewReno

Figure 6: Comparison of Packet Loss Recovery Mechanism when $W^* = 640$ and $B = \frac{1}{4}W^*$

is always one unless packet loss occurs in the congestion avoidance phase. Hence μ of NewReno is the highest among others, as in Fig. 2 (b), (c) and (d).

4.2 The Adverse Effect of NewReno's Packet Loss Recovery Algorithm

In this subsection we explain why $\mu_{NewReno}$ is smaller than μ_{SACK} and even smaller than μ_{Reno} when $B = \frac{1}{4}W^*$, by pointing out that NewReno's packet loss recovery mechanism can be too conservative when a lot of packets are lost in the same window. We choose the case when $W^* = 640$ because $\mu_{NewReno}$ is the smallest.

Fig. 6 shows the packet loss recovery mechanism of Reno and NewReno. When $B = \frac{1}{4}W^*$, NewReno suffers from packet losses whereas there are no packet losses when $B \geqslant \frac{1}{2}W^*$. NewReno estimates the bandwidth-delay product as 626 packets when $W^* = 640$, and exponentially increases its congestion window size until 626. But when it increases congestion window from 256 to 512, the bottleneck buffer overflows because B is only $\frac{1}{4}W^*$, 160. 96 packets are dropped from 0.92795 to 0.94836 second at the same window.

In Fig. 6 (b), it takes about 10 seconds for NewReno to recover packet losses completely, because it retransmits lost segments by one per round-trip time, so μ is almost zero from 1 to 11 second.

But Reno recovers packet losses more quickly. In Fig. 6 (a), Reno suffers packet losses exactly in the same time to NewReno. At 1.05399 and 1.25455 second Reno retransmits two lost packets by *Fast Retransmit* algorithm, but the additional lost segments are not retransmitted until a retransmission timeout occurs. At 1.65482 second retransmission timer expires and Reno starts the slow-start again. During the slow-start Reno recovers lost packets quickly so that at about 3 second Reno retransmits

all dropped packets in the first packet drop period. Note that although Reno suffers an retransmission timeout but needs only 2 seconds to recover packet losses, while NewReno requires 10 seconds for complete recovery despite no retransmission timeout occured.

4.3 The impacts of *B* change

For investigating the influence of increasing B, we define the *Buffer Gain* $BG_k(\alpha)$ in the same W^* where $k \in \{Reno, SACK, NewReno\}$ and $\alpha = \frac{1}{4}, \frac{1}{2}, \frac{3}{4}$ and 1 as follows:

$$BG_k(\alpha) \triangleq \frac{\mu_k \; when \; B = \alpha W^*}{\mu_k \; when \; B = \frac{1}{4}W^*}$$

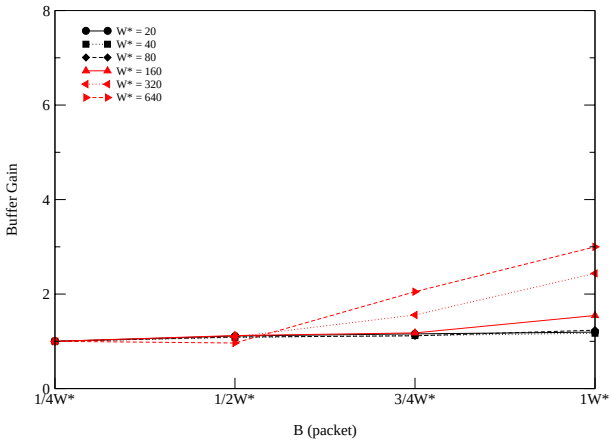
BG_k is the criterion for measuring how μ is improved as buffer increases.

Fig. 7 shows that BG_k generally increases as B increases regardless of TCP versions. Note that when $W^* \leqslant 80$, BG_k is almost identical in Fig. 7 (a), (b) and (c). That means when BDP is relatively small, increasing the size of the bottleneck buffer does not increase μ significantly in all versions. But BG_k relatively increase highly when BDP increases, especially $W^* \geqslant 320$ in all versions.

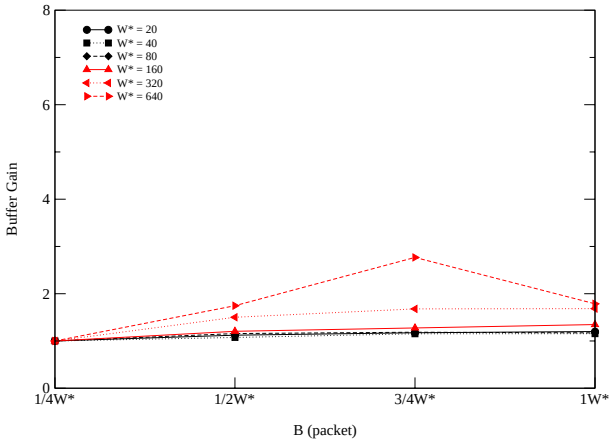
In Reno, We can see that increasing B from $\frac{1}{2}W^*$ to higher value causes relatively significant improvement of μ when $W^* \geqslant 320$, because BG_{Reno} is higher than 2 when $B \geqslant \frac{3}{4}W^*$. Increasing B from $\frac{1}{4}W^*$ to $\frac{1}{2}W^*$ does not cause the improvement of μ in all W^*. When $W^* = 160$, μ is slightly improved when $B = W^*$ but otherwise BG_{Reno} is almost one.

BG_{SACK} is almost the same as BG_{Reno} when $W^* \leqslant 160$. When $W^* = 320$, BG_{SACK} increases from 1 to 1.5 when B changes $\frac{1}{4}W^*$ to $\frac{1}{2}W^*$, but the additional increasing of B does not increase BG_{SACK} significantly. When $W^* = 640$, BG_{SACK} increases linearly until $B = \frac{3}{4}W^*$. The slope of BG_{SACK} is the highest in that case. But $BG_{SACK}(1)$ is smaller than $BG_{SACK}(\frac{3}{4})$ because SACK suffers a retransmission timeout when $B = W^*$ and $W^* = 640$ while there are no timeouts when $B = \frac{3}{4}W^*$.

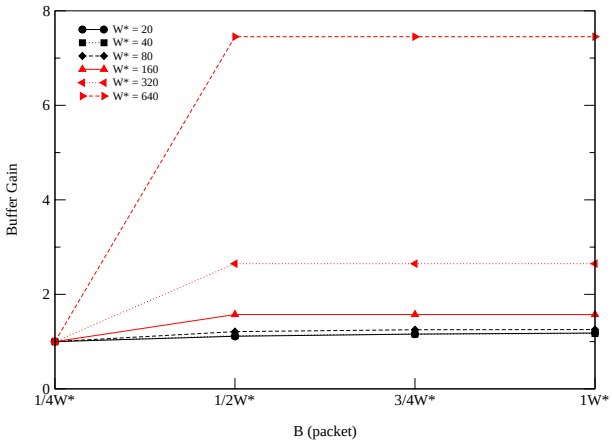
Fig. 7 (c) shows that $\mu_{NewReno}$ significantly increases when B increases from $\frac{1}{4}W^*$ to $\frac{1}{2}W^*$ especially when BDP is relatively large, because $BG_{NewReno}(\frac{1}{2})$ is 2.2 times higher than $BG_{NewReno}(\frac{1}{4})$ when $W^* = 320$ and 7.4 times higher than $BG_{NewReno}(\frac{1}{4})$ when $W^* = 640$. Note that when $B \geqslant \frac{1}{2}W^*$, there is no improvement of μ although B increases because NewReno does not suffer from packet losses during the slow-start phase when $B \geqslant \frac{1}{2}W^*$.



(a) Reno



(b) SACK



(c) NewReno

Figure 7: Comparison of *Buffer Gain*

5 Conclusion

In this paper, we compare the performance of TCP Reno, NewReno, and SACK as the bandwidth-delay product increases. From the simulation results, we obtain the following key results:

- As BDP increases, the average link utilization decreases regardless of TCP version and bottleneck buffer size.
- NewReno always outperforms Reno and SACK if packet losses do not occur during the slow-start phase.
- Increasing the bottleneck buffer size can improve the link utilization, especially when B is larger than $\frac{3}{4}BDP$ in Reno and SACK and larger than $\frac{1}{2}BDP$ in NewReno.

We explain the reason why the average link utlization decreases as BDP increases. We also point out that the average link utilization of NewReno can degrade significantly when packet losses occurs during the slow-start phase because its packet loss recovery mechanism.

References

[1] V. Jacobson, R. Braden, D. Borman, *"TCP Extensions for High Performance,"* RFC 1323.

[2] V. Jacobson, *"Congestion Avoidance and Control,"* In Proceedings of ACM SIGCOMM 1988.

[3] V. Jacobson, *"Modified TCP Congestion Avoidance Algorithm,"* message to end2end-interest mailing list, April, 1990. ftp://ftp.ee.lbl.gov/email/vanj.90apr30.txt

[4] K. Fall, S. Floyd, *"Simulation-based Comparisons of Tahoe, Reno, and SACK TCP,"* Computer Communications Review, July 1996.

[5] C. P. Charalambous, V. S. Frost, J. B. Evans, *"Performance Evaluation of TCP Extensions on ATM over High Bandwidth Delay Products Networks,"* IEEE Communications Magazine, July 1999.

[6] T. V. Lakshman, U. Madhow, *"The Performance of TCP/IP for Networks with High Bandwidth-Delay Products and Random Loss,"* IEEE/ACM Transactions On Networking, Vol. 5, No. 3, June 1997.

[7] L. Brakmo, L. Peterson, *"TCP Vegas: End-to-End Congestion Control in a Global Internet,"* IEEE Journal on Selected Areas in Communications, 13(8):1465-1480, October 1995.

[8] S. Keshav, *"A Control-Theoretic Approach to Flow Control,"* In Proceedings of ACM SIGCOMM 1991.

[9] V. Paxson, *"End-to-End Internet Packet Dynamics,"* In Proceedings of ACM SIGCOMM 1997.

[10] J. C. Hoe, *"Improving the Start-up Behavior of a Congestion Control Scheme for TCP,"* In Proceedings of ACM SIGCOMM 1996.

[11] C. Barakat, E. Altman, W. Dabbous, *"On TCP Performance in a Heterogeneous Network: A Survey,"* IEEE Communications Magazine, January 2000.

[12] J. Padhye, S. Floyd, *"Identifying the TCP Behavior of Web Servers,"* Preliminary Draft, July 2000. Available at http://www.aciri.org/tbit/tbit.ps

[13] Soo-hyeong Lee, Haewon Lee, Yanghee Choi, *"Modeling and Analysis of End-to-end Window-based Congestion Control and Its Application to TCP-Vegas,"* submitted to IEEE INFOCOM 2001.

[14] S. McCanne and S. Floyd, *ns(network simulator)*, version 2, http://www-mash.cs.berkeley.edu/ns