TCP Implementations and False Time Out Detection in OBS Networks

Xiang Yu[†], Chunming Qiao[†] and Yong Liu[*]
[†]Department of Computer Science and Engineering, State University of New York at Buffalo
[*]Department of Computer Science, University of Massachusetts at Amherst

***Abstract*— This paper compares Reno, New-Reno and Selective Acknowledgements (SACK), the three most common TCP implementations today in (future) optical burst switched (OBS) networks. In general, SACK, which considers multiple Triple Duplicated ACKed (TD) losses in one round, is found to perform best in OBS networks, while New-Reno, which improves Reno in packet switched networks by fast retransmission in responding to partial ACKs, may however perform *worse* than Reno.**

All three TCP implementations react to a Time Out (TO) loss in the same way (i.e, using Slow Start). In OBS networks, where a burst may contain all packets from one round, and a burst loss occurs mainly due to contention instead of buffer overflow, such a TO event may no longer imply heavy congestion, or in other words, it may be a false TO or FTO. Such FTOs, which may be common in OBS networks especially for fast TCP flows, can significantly degrade the performance of all existing TCP implementations. Accordingly, we also propose a new TCP implementation called Burst TCP (BTCP) which can detect FTOs and react properly, and as a result, improve over the existing TCP implementations significantly.

I. Introduction

Optical Burst Switching (OBS) (see [9], [10] for example) is a promising technology which integrate IP and Wavelength Division Multiplexing (WDM) and is expected to support the future Internet backbone with huge bandwidth demand. Since TCP [11] is the prevailing mechanism for data transmission today, and its variations will likely remain dominant in the next generation Optical Internet based on OBS, understanding the performance of the current TCP implementations in OBS network becomes an important issue.

In a TCP/IP over OBS network, the TCP sender/receiver is connected to an OBS network through several IP routers, which form two local IP access networks, hereafter called sender-side and receiver-side respectively. It is noted that both the local IP access networks and the OBS network can have loss (packet loss vs burst loss). The impact of packet losses on the performance of current TCP implementations have been studied in many previous works such as [3], [6], [7], and in this paper, we will focus on the impact of burst losses in the OBS network.

The basic transmission unit in OBS networks is a burst, which usually contains a number of consecutive packets/segments[1]. Whenever a burst loss occurs, a TCP sender will be notified with the loss of a number of consecutive packets belonging to the lost burst, either by duplicate/partial ACKs as in Reno and New-Reno, using the information contained in received ACKs as in SACK, or by an expired timers in all three TCP implementations. How such a burst loss affects different TCP implementations has not studied in current literatures as far as we know. For example, most recent studies on TCP over OBS either are based on TCP Reno only as [2] and without considering Reno's fast recovery algorithm, or simply ignored the details of TCP implementations as [1], [4].

[1]We will use the terms packet and segment interchangeably as [3], [6], [7].

For Reno TCP, consecutive packet losses in a burst can shrink the congestion window a consecutive number of times due to multiple TD retransmissions, and may eventually result in TO retransmission. New-Reno TCP improves over Reno in packet switched networks by avoiding TO resulted from multiple TD losses. However, it can have a *worse* performance in OBS networks than Reno when the size of a lost burst (and accordingly, the number of lost packets) is large because New-Reno insists on retransmitting only one lost packet in each retransmission round, and during such a retransmission phase, there can be no new packets transmitted.

Some recently proposed TCP implementations such as Selective Acknowledgements (SACK) [3] and Forward Acknowledgements (FACK) [6] address the inefficiency of Reno and New-Reno in dealing with multiple packet losses in one sending round by adding the information about missing packets in a receiver's window to the ACKs so that a sender can retransmit multiple lost packets in one round. In this paper, we will show that SACK can also achieve a better performance than Reno and New-Reno in OBS networks.

However, all the current TCP implementations deal with a TO loss in the same way by performing slow-start. While such a TO loss indicates serious congestion in a packet-switched network as all the packets in the same round are lost, it does not necessarily indicates serious congestion in an OBS network. For example, for a fast TCP flow, whose arrival rate at an OBS assembly node is large, all the packets from the same sending round can always be assembled in one burst. If such a burst is lost due to contention (among multiple bursts), no ACKs can be sent back, and accordingly, the burst loss will eventually trigger a TO event, even though the burst loss occurs with more or less a random probability, rather than due to buffer overflow induced by serious congestion. In other words, such a TO event will generate a false indication of serious congestion, and accordingly will be called false TO or FTO hereafter. FTOs will unnecessarily force all current TCP implementations to perform Slow Start, and thus result in significant performance degradation.

This paper makes two major contributions. First, we use simulations to evaluate the performances of Reno, New-Reno and SACK in OBS networks, and in particular, demonstrate their inefficiencies in dealing with FTOs. Another major contribution is that we propose a new implementation called Burst TCP (BTCP) which can detect FTOs and react properly. More specifically, upon detecting a FTO, BTCP treats the loss as a TD loss, and enters the fast retransmission phase; If BTCP determines that a TO is a true TO (i.e., not a FTO) resulted from several (packet/burst) losses, or there have been several consecutive FTOs (which also indicate serious congestion in the OBS networks), BTCP performs the usually timeout retransmissions with Slow Start as all current TCP implementations do. In this paper, we will describe three FTO detection methods, and discuss their tradeoffs between FTO detection accuracy and implementation complexity. We will show through both analysis and simulations that BTCP can have a much higher throughput than existing TCP implementations especially for fast TCP flows while remaining to be as responsive to serious congestion.

The rest of the paper is organized as follows. Section II introduces the background for OBS networks and the current Reno, New-Reno and SACK implementations. Section III evaluates the performances of Reno, New-Reno and SACK in the OBS networks via NS-2 simulations. The proposed BTCP with FTO detection methods is described in Section IV, and its performance is evaluated in Section V. Finally, Section VI concludes this paper.

II. BACKGROUNDS

Two main characteristics of OBS that result in different TCP performance from (electronically) packet-switched networks are burst assembly at OBS ingress nodes, and bufferless switching within the OBS core. More specifically, bufferless switching implies that burst losses within OBS networks could be more or less random due to contention among multiple bursts instead of buffer overflow. On the other hand, burst assembly affects the TCP performance mostly in two ways: delay penalty due to an increased TCP round trip time, and correlation gain from being able to send more packets between two loss events [2], [12]. They both depend on the burst assembly time T_b, and offset each other to some extent. While the delay penalty affects different TCP implementations in more or less the same way[2], the correlation gain, which is the net effect of the so-called delayed first loss (DFL) gain and the retransmission penalty [12], could be different for different TCP implementations with different retransmission mechanisms.

A. Slow, Fast and Medium-rate TCP Flows in OBS networks

In this paper, we approximate each of the local IP access network as just one link with a constant access bandwidth λ (which is also the TCP sending/receiving rate). Below, we will review three typical TCP traffic scenarios in OBS networks: slow TCP flow, fast TCP flow and medium-rate TCP flow.

[2]Here, we assume a simple time-based burst assembly algorithm where T_b is fixed.

1) (Extremely) Slow TCP Flow: In this traffic scenario, a TCP flow has an extremely slow arrival rate λ or equivalent, an extremely small assembly time, T_b which satisfies

$$\lambda \cdot T_b < 1 \tag{1}$$

As a result, each burst contains only one packet from the TCP flow. In addition, since the extra delay introduced by burst assembly is small compared to the Round Trip Time (RTT) for the slow TCP flow, the TCP performance in an OBS network approximates to that in an electronic packet-switched network whose packet loss rate is the same as the given burst loss rate. Hereafter, we will ignore this case as the results from packet-switched networks should apply.

2) Medium Rate TCP Flow: In this traffic scenario, a TCP flow has a medium arrival rate (and a medium assembly time) so that one burst contains more than one packet from the TCP flow, but not all the packets in one sending round. In other words, we have

$$1 \leq \lambda \cdot T_b < W_m \tag{2}$$

Usually, there will be more than one bursts transmitted in the OBS network for each sending round, with each burst containing more than one TCP packets.

3) Fast TCP Flow: In this traffic scenario, the sending rate λ of a TCP flow is so fast (or the assembly time T_b is so large) that all packets sent in one round (even after the sending window reaches its maximum size of W_m) will arrive within the same assembly period, and thus be assembled into one burst. In other words, we have

$$\lambda \cdot T_b \geq W_m - 1 \tag{3}$$

For such a flow, there is always one burst that contains all TCP packets transmitted in the OBS network for each sending round.

In this paper, we will focus on medium and fast TCP flows in OBS networks, which is of more practical interest, especially with high-speed access networks and OBS networks with a reasonably large assembly time. For example, assume that the access bandwidth is 2.5Gbps in the local IP access networks, with each TCP packet size of 1KB, and the maximum window size of 2Mb, any assembly time larger than $2Mb/2.5Gbps = 0.8ms$ would make a TCP flow a fast TCP flow. In OBS networks, the assembly time could be a few ms long.

B. Reno, New-Reno, and SACK TCP Implementations

Reno TCP refers to TCP with Slow Start, Congestion Avoidance, Fast Retransmit and Fast Recovery algorithms. When Reno starts, it enters the Slow-Start phase first with a congestion window of size 1, and then exponential expands its sending window after all packets transmitted in the previous round are acknowledged. When the congestion window reaches a certain threshold, Reno enters the Congestion Avoidance phase during which the window expanding speed slows down, that is, Reno increases the congestion window size by 1 packet after all packets from the previous round are acknowledged.

Reno distinguishes two kind of losses, namely timeout (TO) losses and triple duplicate (TD) losses. A TD loss is assumed when a Reno sender receives three duplicate ACKs for the same packet, and the sender will not wait for a TO before retransmitting the lost packet. During the retransmission, the sender halves its congestion window in response to the congestion indication by the loss. The rationale behind this is that a TD loss only indicates a light congestion. On the other hand, after a TO loss, where no more than 3 packets are successfully transmitted before the timer expires (which usually indicates a heavy congestion), Reno enters the Slow Start phase, followed by the Congestion Avoidance phase, to retransmit the lost packets as well as new packets. Note that when multiple packets are lost in the same round, e.g., when a burst containing a large number of packets is lost, Reno will halve its congestion window every time it successfully retransmits one lost packet and receives three duplicate ACKs for the next lost packet in the burst, and eventually, its size may become less than 3 (e.g., this will be the case if the congestion window at the time burst loss occured is small enough). After that, since it is impossible to receive three duplicate ACKs, a TO event may be triggered, which will bring Reno to the Slow-Start phase.

New-Reno makes the following small change to Reno. Even when multiple packets from a single window of data are lost, New-Reno can recover without a TO by retransmitting one lost packet per RTT upon receiving each partial ACK, without waiting for three duplicate ACks, and does not halve the congestion window until all the lost packets from that window have been retransmitted. With the above changes, New-Reno can improve the performance in packet switched networks. However, in OBS networks, with a large burst lost, New-Reno can prolong the retransmission period significantly during which no new packets can be sent which may decrease its performance.

The congestion control algorithm implemented in SACK is a conservative extension of Reno's congestion control, in that it uses the same algorithms for increasing and decreasing the congestion window. The difference is that the option field in SACK contains a number of SACK blocks, where each SACK block reports a non-contiguous set of data that has been received and queued. With the block information in ACK, the TCP sender will be able to send more than one lost packets at a time, which helps improve the TCP performance in OBS networks.

III. Performance Comparison Among Existing TCP Implementations

In this section, we compare the performance of Reno, New-Reno and SACK in OBS networks using NS-2 simulation.Previous work has evaluated the performance of these TCP implementations with a few packet losses within a sending round, but not as many packet losses as what occurs in an OBS network with a burst loss.

Generally speaking, all three TCP implementations have the same Slow Start and Congestion Avoidance algorithms, and the DFL gain and delay penalty mentioned earlier will be the same as long as the assembly time or burst size is kept the same. The differences between various TCP implementations come from the fast retransmission and fast recovery mechanisms, and their interactions with burst assembly in OBS networks, which are the focus of this section.

In our simulation, the TCP sender and receiver connect to OBS edge nodes with a link whose propagation delay is 10ms. The OBS network is modelled with two edge nodes, and two core codes which form a path of four nodes using three fiber links, each having 10ms delay and 10Gbps bandwidth. Therefore, the round trip time (RTT) without including the assembly time is around $2 \times 50ms$ or 0.1s. By default, the packet size is fixed at 1KB, and the maximal window limitation varies from 10KB to 200KB. The results from the simulations were generated by tracing packets inside bursts departing the OBS ingress node. For each graph, the X-axis shows the bursts' departure time in seconds, the Y-axis shows the packets' number mod 60.

A. With One Burst Loss

This section highlights the performance differences between Reno, New-Reno and SACK in OBS networks with one burst loss.

1) Medium Rate TCP flows: To simulate a medium rate flow, we assume that the access bandwidth is 125KBps.

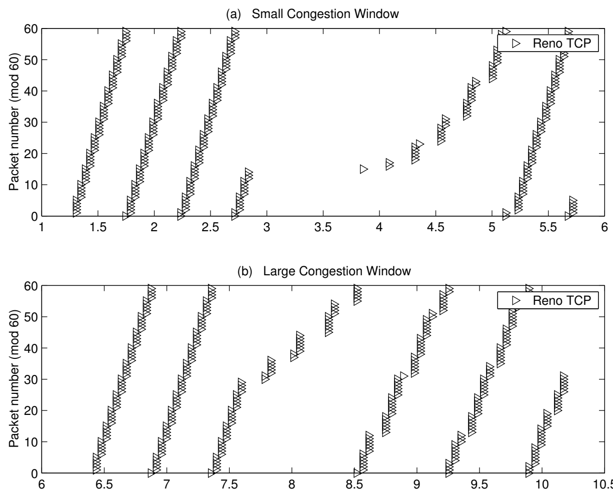
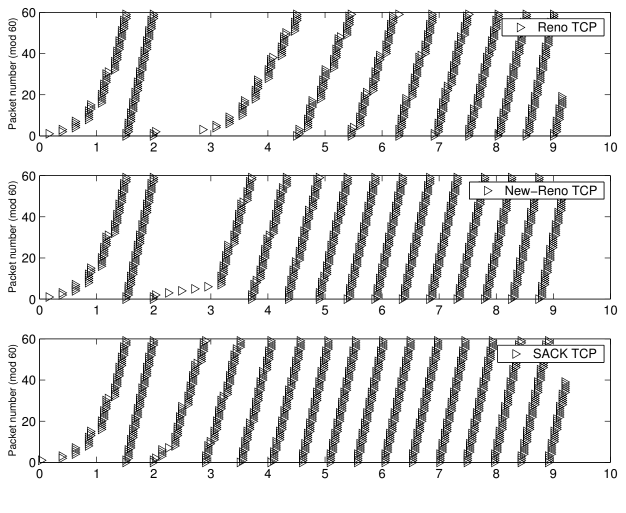


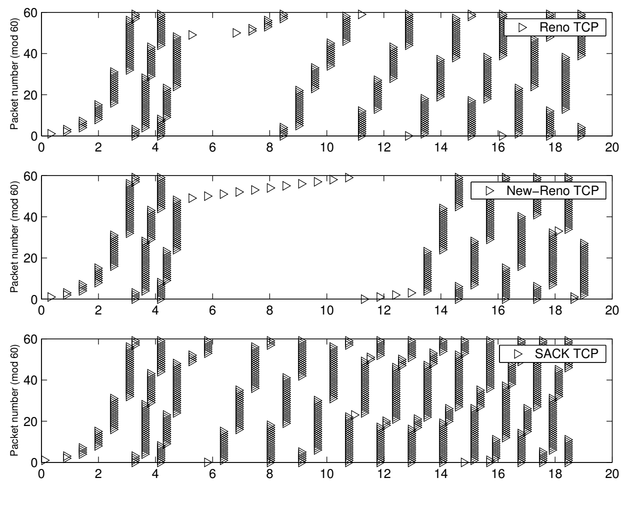
Fig. 1. Packet trace with one burst loss occurred at (a) $W = 20$ and (b) $W = 200$ for a Reno TCP flow ($T_b = 0.04s$)

Figure 1 illustrates the impact of the congestion window size on Reno's TD loss event, where the length of the lost burst is $BL = 5$ packets (which is the maximal value since $0.04 \times 125 = 5$). As we can see from Figure 1 (a), if the congestion window[3] is small, i.e., $W = 20$ packets, at the time when the burst is lost, the window size will be reduced to less than 3 at time 3 after three retransmission rounds but before all 5 packets lost in a burst can be retransmitted. Therefore, without being able to receive three duplicate ACKs any longer, a TO occurs at time 3.8 and a Slow Start phase begins. However, in Figure 1(b) where the congestion window size has already grown to reach the maximal limit of 200 packets by the time the burst loss occurs, Reno can recover from retransmission

[3]Note here with a relatively small RTT, TCP packets are pipelined and congestion window cannot be represented in the graph



(a) $T_b = 0.04$



(b) $T_b = 0.2$

Fig. 2. Packet traces with one burst loss ($W = 40$ when a burst is lost)

stage and re-enter the congestion avoidance phase without a TO event.

Figure 2(a) compares different TCP implementations upon one burst loss when both BL and W are small (5 and 40 in the simulation, respectively). It can be seen that SACK has the best performance because the ACK can indicate the block of packets lost, and the sender sends out the all the lost packet again upon receiving the ACK. In addition, New-Reno generally performs better than Reno because New-Reno detects the loss of next packets upon receiving a partial ACK, without waiting for 3 duplicate ACKs or a TO (while Reno will have a TO as discussed earlier). In addition, New-Reno's congestion window will only be halved once after recovered (i.e., $W = 40/2 = 20$), which makes New-Reno can recover quickly from the loss of a small burst.

Figure 2(b) shows that when the lost burst length is large, ($BL = 15$ in simulation) while W is still relatively small when the burst loss occurs (e.g., 40), New-Reno performs worse than Reno. This is because New-Reno only retransmits one lost packet in one round, and hence needs a long time to finish all 15 retransmissions during which no new packets can be sent. Reno, on the other hand, will have a TO as before, but new packets may be transmitted before the TO and in addition, the TO value is much smaller than 15 RTT in New Reno's fast retransmission phase, and Reno's transmission after TO is much more efficient because it exponentially increases the sending window size. Also from Figure 2(b), we can see that in this case, SACK has a much better performance than Reno and New-Reno due to its selective acknowledgements.

We note that if BL is small but W at the time burst loss occurs is relatively large as in Fig. 2(b), Reno will not have a TO, and hence its performance will be comparable to that of New-Reno.

2) Fast TCP Flows: To simulate a fast TCP flow, we assume that the access bandwidth is 1MBps, and the burst assembly time is $T_b = 0.2s$. In this case, since there is only one burst containing all the packets in one round, a burst loss will trigger a timeout (TO) event in Reno, New-Reno and SACK, which have exactly the same performance as they all use Slow Start for retransmission after a TO loss as shown in Figure 3.

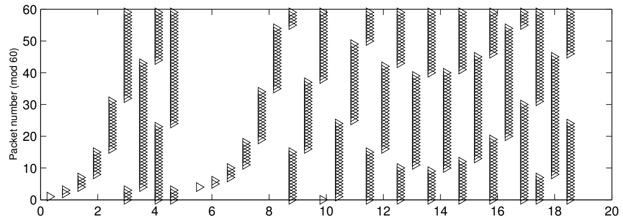


Fig. 3. Packet trace with one burst dropping for a fast TCP flow

B. Multiple Burst Losses

In this section, we compare the performances of the three TCP implementations with multiple burst losses. For a fast TCP flow, since all three TCP implementations apply the same timeout retransmission mechanism every time a burst is lost, they are expected to have the same performance with multiple burst losses as well. Therefore, we only consider multiple burst losses in a medium rate TCP flow.

Figure 4(a) illustrates the performances of the three different TCP implementations with a medium-high loss rate. When losses are scattered before time 12, the three TCP implementations have much different performances due to different TD retransmissions, and when losses occur one after another immediately, the three TCP implementations have the same performance as multiple back to back burst losses will easily trigger a TO event (after which, all TCP implementations perform the same timeout retransmissions) as shown after time 28 in Reno, after time 21 in New-Reno and after time 16 for SACK in Figure 4(a). On the other hand, Figure 4(b) shows

TABLE I

THROUGHPUT RATIO OF NEW-RENO AND SACK OVER RENO, $T_b = 0.04s$

$log(p)$	-3	-2.5	-2	-1.5	-1	-0.5
$NewReno/Reno$	1.0	1.1	1.29	1.19	1.03	0.93
$SACK/Reno$	1.0	1.07	1.58	1.4	1.33	1.07

TABLE II

THROUGHPUT RATIO OF NEW-RENO AND SACK OVER RENO, $T_b = 0.2s$

$log(p)$	-3	-2.5	-2	-1.5	-1	-0.5
$NewReno/Reno$	1.0	0.94	0.9	0.76	0.93	1.06
$SACK/Reno$	1.0	1.03	1.02	1.1	1.0	1.0

that with a medium-low loss rate, SACK always has the best performance as most, if not all, losses are TD losses.

Table I and II show the performance ratios of New-Reno over Reno, and SACK over Reno, as the burst loss rate varies from low to high. In general, for a low or high loss rate, all the three TCP implementations tend to have the similar performance. This is because the performance difference for different TCP implementations comes from the TD retransmission stage only. Accordingly, a high burst loss rate usually leads to a higher probability of a TO with which there is no difference in TO retransmissions in the three TCP implementations. On the other hand, a low burst loss rate leads a low TO probability but also a low TD loss probability. Accordingly, there will also be little performance difference between different TCP implementations. However, with a medium-low to medium-high loss rate, the probability of a TD event can be relatively high (compared to a TO event), and accordingly, the performance difference among different TCP implementations will be more obvious, as shown in the middle two columns of Table I and II.

Table I and II also show the effect of the burst assembly time T_b (or burst length) on the performance ratios of New-Reno over Reno, and SACK over Reno, respectively, It is interesting to note that New-Reno has a better performance for a smaller assembly time (0.04s for possibly smaller burst size, e.g., up to 5), while Reno has better performance for a larger assembly time (0.2s for possibly smaller bust size, e.g., up to 25) as illustrated in Table I and II, respectively. In general, their relative performances depend on the relationship between TCP timeout value RTO, round trip time RTT and the number of packets contained in the lost burst BL. When RTO is much larger than $BL \times RTT$, New-Reno has a better performance than Reno, otherwise Reno has a better performance because the interval between the time that the previous burst lost and the time for next new packet to retransmit is approximately RTO for Reno and $BL \times RTT$ for New-Reno. Since RTT and RTO is almost fixed after TCP starts for some time, BL usually decides the relative performances of Reno and New-Reno. Also note that, when assembly time T_b (or burst length) increases, SACK is still better than Reno but its advantage diminishes due to the fact that TO can happen more easily as fewer burst losses can trigger a TO event, where retransmission mechanisms for all TCP implementations are the same.

For medium loss rates, the differences between their performances are the highest since TD loss happens most often and affect the total throughput more. With different burst length or assembly time, Reno and New-Reno has tradeoffs in their performances.

IV. BURST TCP (BTCP) WITH FALSE TIMEOUT DETECTION

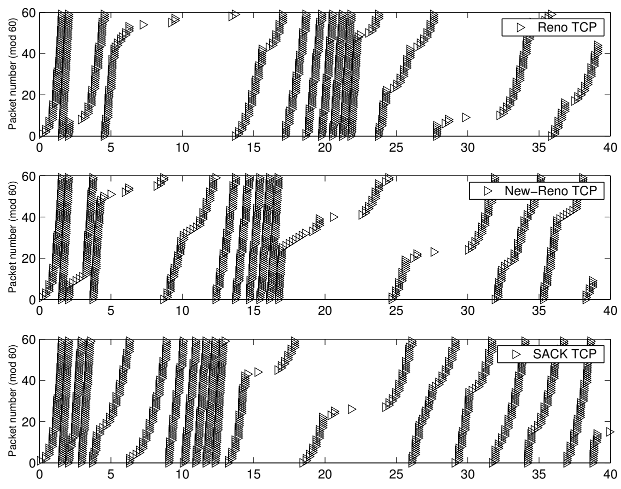
All current TCP implementations assume that a TO event triggered by multiple packet losses indicates a serious congestion in the network. This assumption is valid in (wired) packet-switched networks because such multiple packet losses are resulted from buffer overflows. However, it is not always valid in OBS networks where one burst containing all packets in one sending round may be lost due to contention, since multiple packet losses are due to a single burst loss event, and such a burst loss event is more or less random, and therefore, does not necessarily indicate serious congestions in OBS networks.

We call a TO event a false TO (FTO) if it is caused by a single burst loss in a OBS network which is not in serious congestion. Such FTOs, which can occur quite often for a fast TCP flow in OBS networks, can degrade the performance of all current TCP implementations by forcing them to perform Slow Start.

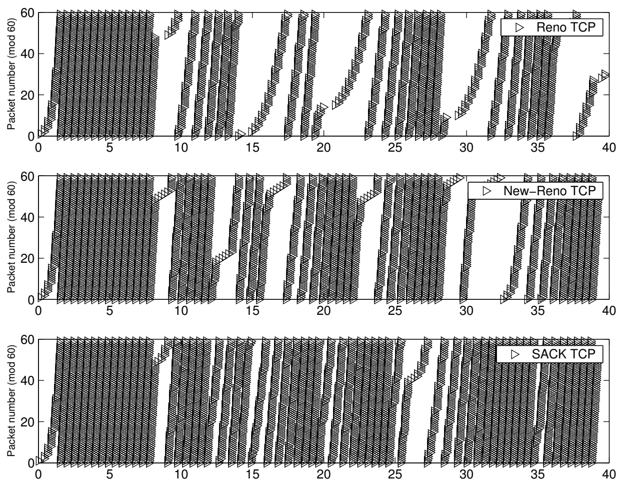
To improve the inefficiency of current TCP implementations in dealing with FTOs, we propose a new TCP implementatin called Burst TCP or BTCP, whose aim is to detect FTOs, and treat them as TD losses, which is the way these packet losses should be treated. In this section, we describe three FTO detection methods that may be used by BTCP.

In order to determine whether a TO event is a FTO or not, a BTCP sender needs to know if this TO is caused by multiple packet losses within either or both of the two IP access networks, called sender-side access network and receiver-side access network. If so, then the TO is a true TO, i.e., not a FTO. Otherwise, the multiple packet losses may be caused by either a single burst loss or multiple burst losses within the OBS newtork. In the former case, the TO is a FTO but in the latter case, it is also a true TO.

The first FTO detection method we propose is for a BTCP sender to estimate the maximal number of packets that can be assembled in a burst. Such a method, called burst length estimation or BLE for short, does not require any changes to OBS networks, and is relatively simple to implement; The second method we propose is to let OBS edge nodes send burst ACK (BACK) to the BTCP sender, which contains the information of the packets contained in a burst arriving at an ingress and/or egress nodes. This BACK based method therefore requires that the OBS edge nodes be able to process TCP packets and send BACKs to the BTCP sender, but can achieve a better FTO detection accuracy than the first method BLE. Last but not least, the third method lets a core OBS node at which a burst has to be dropped send the information of the packets contained in a dropped burst using a burst NAK (BNAK) to the BTCP senders, which requires OBS core nodes to be able to handle TCP packet processing and NAK sending. This BNAK based method can not only achieves the highest FTO detection accuracy among the three FTO detection methods, but also allows the BTCP sender to start TD retransmissions even before a FTO occurs. On the



(a) $p = 0.1$



(b) $p = 0.01$

Fig. 4. Packet traces with multiple burst losses for a medium-rate TCP flow

other hand, the best performance is achieved with the highest implementation complexity among the three FTO detection methods. In the following subsections, we will describe these methods in more detail.

A. FTO Detection using Burst Length Estimation or BLE

To implement the proposed BLE method, the BTCP sender needs to keep one piece of additional state information, which is the burst window size ($burst\_wd$), to estimate the current number of TCP packets contained in one burst in the OBS network. When a TO event happens, the sender compares the current congestion window with the burst window to decide whether it is a FTO or not, and then takes appropriate actions. The following are some of the implementation details:

First, we estimate the burst window size $burst\_wd$ using the following steps:

1. When a new BTCP flow starts to transmit, it resets $burst\_wd$ to zero, then enters the Slow Start phase. The BTCP sender does not need to know about the assembly algorithm and in particular the assembly time used by the OBS network.

2. If the first loss is a TD loss, it means that the current congestion window $cwnd$ is larger than the maximum number of packets contained in a burst. TCP sender updates the $burst\_wd$ as half of the current congestion window size $cwnd$, i.e., $burst\_wd = cwnd/2$. Otherwise if the first loss is a TO loss, it is very possible that the sender's congestion window $cwnd$ is smaller than the number of packets that have been assembled in a lost burst, or in other words, the TO is a FTO. In this case, we set the burst window size as the current congestion window size: $burst\_wd = cwnd$.

3. If a loss is not the first loss, and it is a TD loss, then we set the burst window size as minimum of current burst window size and half of the congestion window: $burst\_wd = \min\{burst\_wd, \frac{cwnd}{2}\}$.

4. If a loss is not the first loss, and it is a TO loss, then there are two subcases: (1) if the TO loss is the first TO loss, then set the burst window size as $\min\{cwnd, burst\_wd\}$; (2) if the TO loss is not the first TO loss, then set the burst window size as $\max\{cwnd, burst\_wd\}$.

To determine if a TO is FTO based on the estimated $burst\_wd$:

1. For any TO loss, as long as $cwnd \leq burst\_wd$ and $burst\_wd > 3$, the sender treats the TO as a FTO, and halves its congestion window and starts fast retransmission of all packets lost.

2. Otherwise, for a TO loss with $cwnd > burst\_wd$ or $burst\_wd \leq 3$, the sender treats the TO as a true TO event, and performs the same TO retransmissions as other TCP implementations do.

Note that the above BLE method requires no coordination or information exchange between the BTCP sender and the OBS nodes. In addition, the method is simple to implement, and can improve the throughput performance when compared to current TCP implementations whose performances are plagued by FTOs.

However, the estimated $burst\_wd$ may not be accurate. Accordingly, it is possible that a true TO will be taken as a FTO by mistake when using the above BLE method for medium rate TCP flows. In addition, even if it is accurate, the above BLE method cannot distinguish multiple packet losses within either or both of the local access network (which are possible albeit with a small probability) from a single burst loss within the OBS network. In other words, when BLE declares a TO to be a FTO, it could be a true TO instead. When it is a true TO, TCP should enter the Slow Start phase immediately as it indicates serious congestion. However, BTCP will enter the Slow Start phase only after several such "FTO" events, which makes it less TCP friendly.

B. FTO Detection using Burst ACK or BACK

One possible enhancement to the previous FTO detection method BLE is to enable an OBS ingress node to send information of the TCP packets in a burst back to the BTCP sender via an Burst ACK or BACK packet. In this way, the sender no longer needs to guess which packets are assembled in the same burst or not. When a TO occurs, the sender will know from the BACK whether all the packets are in the same burst. If so, the TO is very likely a FTO (it is still possible however, that all the packets are lost in the receiver-side access network, but the sender has no way of knowing for sure one way or the other). Otherwise, it is definitely a true TO. Of course, if the BTCP sender did not receive a BACK and a TO occurs, it is safely assume that it is a true TO (since either the multiple packets sent earlier or the BACK packet is lost in the sender-side access network).

The algorithm based on BACK from the ingress node is illustrated as follows:

1. When a burst is created and leaves the ingress node, ingress node collects the sequence number and TCP sender's information for all TCP packets in the burst, and sends an BACK packet to each BTCP sender with the sequence numbers.

2. When the timer expires, a BTCP sender checks the most recent BACK to see if all the lost packets are in the same burst: (1) If they are, then treats the timeout as a FTO and follows the normal TD retransmission as described above. (2) Otherwise, (or if the packet sequence numbers cannot be found in any BACK packets), treats the TO as a true TO and starts retransmission using Slow Start.

Note that one may also ask an OBS egress node to send a BACK packet *instead*. In such a case, when a BACK packet is received, the sender will know that any losses must have occurred within the receiver-side access network. However, if no BACK is received, then the sender cannot tell multiple packet losses within the sender-side access network from a burst loss within the OBS network.

To help distinguish a burst loss within the OBS network from multiple packet losses within either or both of the access networks, one can let both ingress and egress nodes send a BACK packet. In this way, the sender can detect FTOs with accuracy, i.e., BTCP sender knows a TO is a FTO when only a BACK containing all packets from ingress is received but without BACK from egress node, and knows it is a true TO otherwise. Hence, BTCP using BACK based FTO detection can achieve *better throughput performance* than the current TCP implementations while being as responsive to serious congestion within the IP access networks as the current TCP implementations.

Note that, the OBS ingress and egress nodes have proper electronic interfaces to IP routers on one side as well as electronic memory to assemble and disassemble packets into and from a burst, respectively. Nevertheless, they are required by the BACK method to be able to understand TCP and send BACK to BTCP senders.

C. FTO Detection using Burst NAK (BNAK)

In an OBS network, a (OBS) control packet is generated and sent by an ingress node for each (data) burst the ingress node assembles and sends. Such a control packet contains information about the burst (including the routing information), and is processed at each and every core node so as to reserve bandwidth and set up the switching fabric for the burst.

In the proposed FTO detection method using burst NAK or BNAK packets, we can let each control packet contain the information of the (TCP) packets within the corresponding burst. At any core node where the burst runs into contention and has to be dropped, the core node constructs a BNAK packet based on the control packet corresponding to the dropped burst, and sends the BNAK to the BTCP sender.

If the sender receives such a BNAK that contains information on all packets in the $cwnd$, then it knows for certain that any TO associated with a packet mentioned in the BNAK is a FTO. Otherwise, (either no BNAK is received or the BNAK received does not contain information on all packets), then the TO is a true TO.

Based on the above discussion, the BNAK based FTO detection can achieve the same highest accuracy in FTO detection as the BACK based FTO detection where both the ingress and egress nodes send a BACK packet. Moreover, BNAK can result in a much better performance than BACK (although the former requires more complex implementation as well). This is because, as mentioned earlier, with a fast TCP flow, all the packets transmitted in one sending round are assembled into one burst. Accordingly, a TCP sender will be notified of each packet loss (as a part of a burst loss) in OBS networks by a TO event only. As the TCP timeout value is set to be several times larger than its RTT, waiting for the TO to occur before retransmission starts can be quite inefficient. Using the the proposed FTO detection method BNAK, the sender can start retransmission of lost packets as soon as the BNAK is received, which should be less than a RTT and thus much earlier than TOs for these lost packets. Therefore, this method improves the throughput of BACK or BLE based methods further while remaining to be as responsive to serious congestion as the current TCP implementations.

V. Performance of BTCP

In this section, we evaluate the performance of BTCP with different FTO detections via both analysis and simulations. Since BTCP differs from existing TCP implementations mainly in their reactions to FTOs, we will focus on a fast TCP flow for which FTOs are most likely to occur.

A. Analytical Results

As in Sec.III, we will focus on the impact of the loss within the OBS network. More specifically, we assume the number of packets from one TCP flow contained in one burst does not affect the loss probability of the burst. Such an assumption is reasonable given that the size of the burst is determined by the total number of packets from many flows.

1) BTCP using BLE or BACK: If each burst is assumed to have the same independent loss probability p, then the average number of rounds successfully transmitted between two "TO" loses is $\frac{1-p}{p}$ [2]. When the loss rate is not very low, the maximum window limitation is relatively high, i.e., $W_m > 1/p$, and in addition, TO losses are distributed evenly over time, then the sender starts retransmission from W_0 will reach W ($W < W_m$) before the next TO event occurs.

For a fast TCP flow without FTO detection (whether it is Reno, New-Reno or SACK, they has the same performance as discussed in Sec.III), $W_0 = 1$ and $E[W] = \frac{1-p}{p}$, and the TCP sender begins transmitting $W-1$ packets in $log_2(\frac{E[W]}{2})$ rounds during the Slow Start phase, and continues to transmit $W/2$ rounds during the congestion avoidance phase. Therefore, the average total number of packets transmitted between two TO events, denoted by $E[H_1]$, is:

$$\begin{aligned} E[H_1] &= (E[W]-1) + \frac{E[W]}{2} \times (\frac{E[W]}{2} + E[W])/2 \\ &= \frac{3}{8}E[W]^2 + E[W] - 1 \end{aligned} \quad (4)$$

As we can also represent the transmission rounds as

$$\frac{1-p}{p} = log_2(\frac{E[W]}{2}) + \frac{E[W]}{2} \quad (5)$$

we can obtain $E[W]$ in (4) by solving (5), and then obtain $E[H_1]$.

For the fast TCP flow in BTCP with FTO detection based on BLE, we have $W_0 = W/2$, therefore $E[W - W/2] = E[W/2] = \frac{1-p}{p}$ and $E[W] = \frac{2(1-p)}{p}$. The total number of packets transmitted between two TO events, denoted by $E[H_2]$, is:

$$E[H_2] = \frac{1-p}{p} \times (\frac{E[W + \frac{W}{2}]}{2}) = \frac{3(1-p)^2}{2p^2} \quad (6)$$

Note that under the assumption that there is no loss in the local access networks, BTCP using BLE performs the same as BTCP using BACK.

Since the time between two TOs called time out period (or TOP) in BTCP using BLE/BACK (which is $\frac{1-p}{p}RTT + RTO$) are the same as TCP without FTO detection, their throughput ratio is $E[H_2]/E[H_1]$, whose numerical results are illustrated in Figure 5(a) for the case where $W_m p > 1$. The figure also shows the average congestion window size ratio between BTCP and TCP, which is always larger than 1 since BTCP can start with a larger congestion window size W (up to $W_m/2$) after a FTO while TCP will always start at $W = 1$. This increase in the window size is a reason for the increase in the throughput.

When the loss rate is low, and the maximum window limitation is relatively low too, i.e., $W_m < 1/p$, the maximal limit on the congestion window size can be easily reached for most of the time. In such a case, $E[H_1]$ and $E[H_2]$ can be similarly recalculated as:

$$\begin{aligned} E[H_1] &= (W_m - 1) + \frac{3W_m}{8} + W_m(\frac{1}{p} - \frac{W_m}{2} - log_2(\frac{W_m}{2})) \\ &= \frac{W_m}{p} - \frac{W_m^2}{8} + W_m - 1 - W_m log_2(\frac{W_m}{2}) \end{aligned} \quad (7)$$

and

$$E[H_2] = (\frac{1}{p} - \frac{W_m}{2}) \times W_m + \frac{3W_m^2}{8} = \frac{W_m}{p} - \frac{W_m^2}{8} \quad (8)$$

Thus, the throughput ratio becomes:

$$\frac{E[H_2]}{E[H_1]} = 1 + \frac{W_m(log_2\frac{W_m}{2} - 1) + 1}{E[H_1]} \quad (9)$$

which is shown in Figure 5(b). From the Figure, we can see that the benefit of FTO detection in BTCP increases with the maximum window limitation W_m. This is because with a larger W_m, BTCP can start with a larger $W = W_m/2$ after a FTO while current TCP has to start with $W = 1$, similar to the explanation provided for Figure 5 (a). In addition, the large the loss rate p, the better the improvement as more TO events may occur.

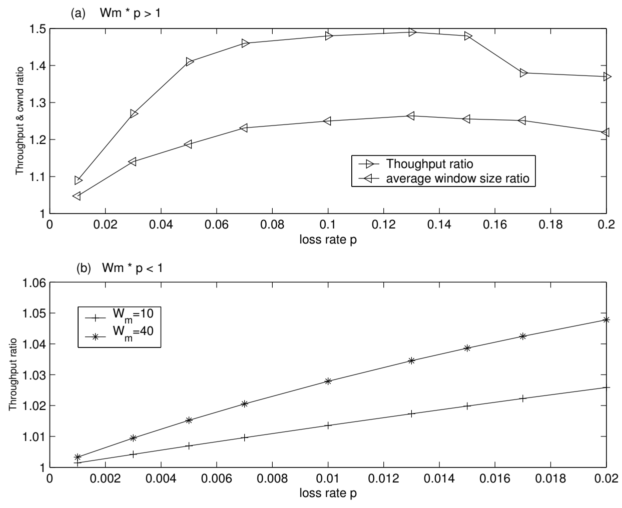


Fig. 5. Analytic results of BTCP vs TCP

2) BTCP using BNAK: For BTCP with FTO detection method based on BNAK, the number of packets transmitted between two TOs is also identical to that by TCP without FTO detection. However, for a FTO, the retransmission round can start as soon as a BNAK is received, which is RTT time after the loss round, instead of RTO time in TCP without FTO detection. That is, the total time between two TOs in BTCP using BNAK is $|TOP_2| = \frac{RTT}{p}$, while the total time between two TOs in TCP without FTO detection is $|TOP_1| = \frac{1-p}{p}RTT + RTO$. Therefore, the throughput enhancement due to BTCP using BNAK when compared to TCP without FTO detection is:

$$\begin{aligned} \frac{E[H_2]/|TOP_2|}{E[H_1]|TOP_1|} &= \frac{E[H_2]}{E[H_1]} \times \frac{\frac{1-p}{p}RTT + RTO}{\frac{RTT}{p}} \\ &= \frac{E[H_2]}{E[H_1]}(1 + \frac{RTO - RTT}{RTT} \times \frac{p}{1-p}) \end{aligned} \quad (10)$$

As $RTO - RTT > 0$, the above analysis also implies that BTCP using BNAK will achieve better performance improvement than BTCP using either BLE or BACK, and such an improvement will increase with p.

Note that, the above analysis in (10) did not consider consecutive FTOs, which are possible when the loss is unevenly distributed and the loss rate is high. Since BTCP with BNAK can also eliminate the exponential backoff time when compared to current TCP implementations (which will treat them as multiple true TOs), the enhancement of BTCP using BNAK according to our analysis in (10), may be an underestimation. In fact, BTCP using BLE/BACK can also eliminate the exponential backoff time, and accordingly, the previous analysis results for BTCP using BLE/BACK, which is shown in Figure 5 (b), may also underestimate its performance enhancement. We will discuss the case with consecutive FTOs in more details later

B. Numerical Results

We highlight the performance enhancement due to BTCP for a fast TCP flow via simulations in this section. As discussed earlier in Figure 3, there is no significant performance differences among Reno, New-Reno and SACK because they deal with TOs in the same way. Accordingly, we will compare BTCP, which enhances Reno with FTO detection, with Reno.

Figure 6 shows the details of how BTCP using BLE deals with TOs. When the first TO occurs at time 11, Reno triggers TO retransmission and Slow Start, while BTCP can detect that it is a FTO because its current burst window size is equal to the congestion window (i.e., $burst\_wd = cwnd = 40$ in this example). Therefore, BTCP treats this TO event as a TD loss and triggers fast recovery, which enhances the performance when compared to Reno. Also note that for consecutive TO losses during time 28 to 55, BTCP can still respond to temporarily high loss situations in the OBS network by halving the congestion window multiple times in consecutive retransmissions and eventually triggering a true TO event. This means that BTCP can not only improve Reno's performance, but also remains to be as responsive to serious congestion in the OBS network as Reno.

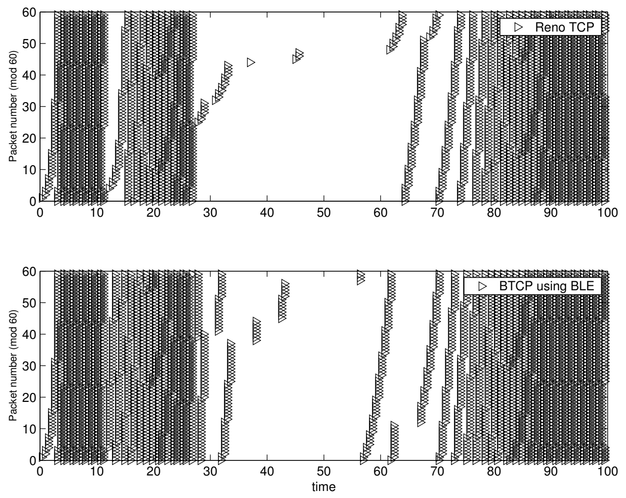


Fig. 6. Packet Trace of TCP and BTCP

1) Throughput Performance: Figure 7 compares the throughput of a single fast TCP flow in an OBS network with fixed loss rate $p = 0.1$, and randomly distributed losses. Here, p is chosen to be relatively large to highlight the performance differences between BTCP using BLE/BACK, BTCP using BNAK and Reno.

As can be seen from the figure, all BTCP versions have a better performance than Reno. While BTCP using BLE and BTCP using BACK have the same performance (since there is no loss in the access networks as discussed in the previous analysis), BTCP using BNAK achieves the best performance since it reduces the waiting time before retransmissions (in response to a FTO) from RTO to RTT.

As mentioned earlier in the analysis (see Fig. 5), the benefit of FTO detection in BTCP increases with the maximum window limitation W_m. This is verified in simulations by comparing Figure 7(b) where $W_m = 20$ with Figure 7(a)

TABLE III
PERFORMANCE ENHANCEMENT OF BTCP VS p ($T_b = 0.04$)

$log(p)$	-3	-2.5	-2	-1.5	-1
$R_{BNAK\text{-}r}$	1.00	1.00	1.05	1.06	1.76
$R_{BLE/BACK\text{-}r}$	1.00	1.00	1.05	1.05	1.17
$R_{BLE/BACK\text{-}e}$	1.00	1.01	1.02	1.05	1.28
$R_{BLE/BACK\text{-}b}$	1.01	1.05	1.07	1.193	1.46

TABLE IV
PERFORMANCE ENHANCEMENT OF BTCP VS T_b ($p = 0.1$)

T_b	0.01	0.02	0.04	0.06	0.08	0.1
$R_{BNAK\text{-}r}$	1.48	1.43	1.35	1.31	1.27	1.16
$R_{BLE/BACK\text{-}r}$	1.20	1.19	1.17	1.18	1.16	1.07
$R_{BLE/BACK\text{-}e}$	1.24	1.24	1.24	1.24	1.24	1.24
$R_{BLE/BACK\text{-}b}$	1.59	1.58	1.46	1.47	1.43	1.43

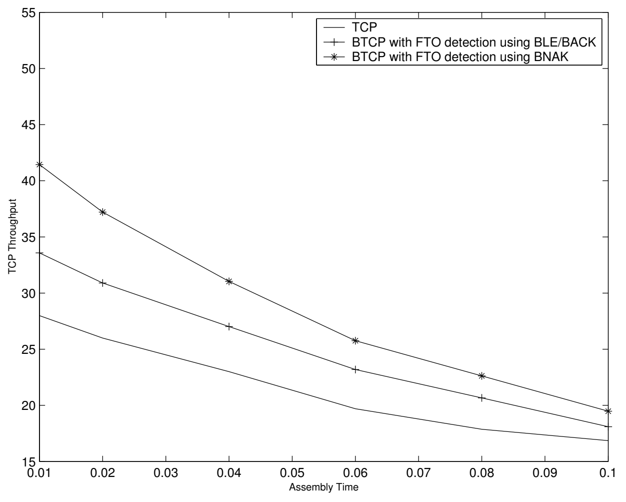
where $W_m = 10$. Note that the throughput of all BTCP versions and Reno decrease with T_b due to the (burst assembly) delay penalty to a fast TCP flow as discussed in [2], [12]. The reason that the throughputs are low when $T_b = 0.01$ and $W_m = 20$ (in Figure 7) (b)) is because the TCP flow simulated no longer qualifies as a fast TCP flow according to (3) (see [12] for throughput analysis of such a medium-fast TCP flow).

With randomly distributed losses, the performance enhancement ratio of BTCP using BNAK over Reno, denoted by $R_{BLE/BACK\text{-}r}$, as well as the enhancement of BTCP using BLE or BACK over Reno, denoted by $R_{BLE/BACK\text{-}r}$, are further illustrated in the first and second rows in Table III and IV for various loss rates and assembly times, respectively. It can be seen from the two tables that both BTCP using BNAK and BTCP using BLE/BACK, especially the former, can be significantly better than TCP when the loss rate is high. Hereafter, we will focus on BTCP using BLE/BACK although most of the discussions apply to BTCP using BNAK as well.

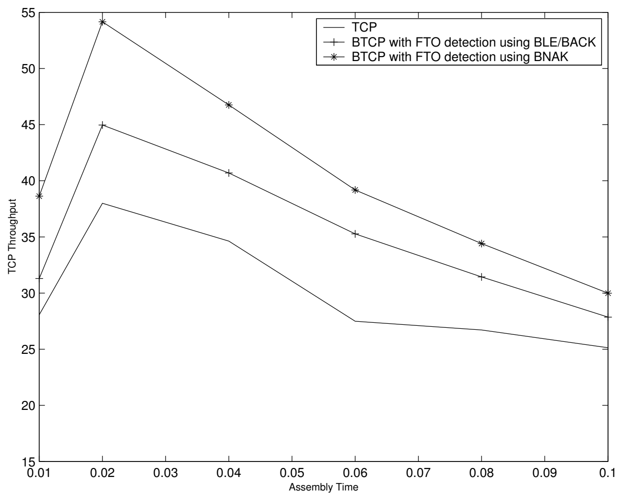
It is noted that the performance enhancement due to BTCP not only depends on the average loss rate as shown in Table III, but also the distribution of the burst losses (Note that for a fast TCP flow, a burst loss is the same as a FTO loss), which can be either randomly distributed as mentioned above, or one of the two additional distributions namely, evenly distributed, or (evenly distributed) batches (e.g., 4 losses occur back to back as a batch, and there is a fixed amount of time between two batches of losses). The last two rows in Tables III and IV show the enhancement ratios with the two additional loss distributions between BTCP using BLE/BACK over Reno, denoted by $R_{BLE/BACK\text{-}e}$ and $R_{BLE/BACK\text{-}b}$, respectively.

As can be seen from Table III, for the same average loss rate, if several burst losses occur back to back in a batch, BTCP can enhance the throughput a lot by eliminating the significant amount of exponential backoff time in TCP due to consecutive TO events. However, if the burst losses occur randomly (but not back to back) or evenly, there is no exponential backoff in TCP, and in addition, when these losses occur closely (but not consecutively), the congestion window can hardly increase in BTCP. This is why $R_{BLE/BACK\text{-}r}$ is not as high as $R_{BLE/BACK\text{-}b}$.

It is also noted that the performance enhancement due to BTCP depends on the assembly time as shown in Table IV (as well as in Figure 7) where $p = 0.1$. To explain the results



(a) $W_m = 10$



(b) $W_m = 20$

Fig. 7. Simulation Comparison of BTCP and TCP

TABLE V
PERFORMANCE OF BTCP WITH TO IN ACCESS NETWORKS

T_b	0.01	0.02	0.04	0.06	0.08	0.1
R_t	1.1059	1.0696	1.1197	1.0450	1.0526	0.9940
R_l	1.0203	1.0160	1.0363	1.0133	1.0189	1.0180

in the table, consider the case with batch losses first. On one hand, TCP will go through several exponential backoff periods as mentioned earlier according to it timeout value RTO, which is independent of the assembly time T_b. On the other hand, BTCP will enter the fast retransmission phase and then eventually the Slow Start phase (due to a true TO triggered by several FTOs). Assume that the length of the batch loss period is fixed as $\alpha_b \times RTO$ and $\alpha_t \times RTO$ for BTCP and TCP respectively, where α_b and α_t are two constants and we should have $\alpha_b < \alpha_t$. Also suppose that $RTT = RTT_0 + T_b$, where RTT_0 is the round trip time without burst assembly, and the total data transmitted in the N rounds (which is fixed for evenly distributed batch losses) before and within the lost batch is W_b for BTCP and W_t for TCP, then the performance enhancement for batch losses is calculated as

$$R_{BLE/BACK\text{-}b} = \frac{W_b/[\alpha_b \times RTO + (RTT_0 + T_b) \times N]}{W_t/[\alpha_t \times RTO + (RTT_0 + T_b) \times N]}$$

$$= \frac{W_b}{W_t}[1 + \frac{(\alpha_b - \alpha_t)RTO}{\alpha_b \times TO + (RTT_0 + T_b) \times N}] \qquad (11)$$

Since all other parameters are treated as constant and $\alpha_t - \alpha_b > 0$, $R_{BLE/BACK\text{-}b}$ in (11) decreases with T_b. For evenly distributed losses, $\alpha_t = \alpha_b$ (because there is no exponential backoffs in both BTCP and TCP), and thus the corresponding enhancement $R_{BLE/BACK\text{-}e}$ is insensitive to T_b. Finally, since the case with random losses is in between the case with batch losses and evenly distributed losses, the corresponding enhancement $R_{BLE/BACK\text{-}r}$ also decreases with T_b.

2) Responsiveness to Congestion: When a (true) TO occurs due to multiple packet losses in either or both of the IP access networks, BTCP using BLE may treat the true TO as a FTO by mistake because it cannot distinguish losses within the OBS network from those within the IP access networks. As a result, BTCP may not respond to serious congestion as effectively as Reno, and hence the loss rate in the network will be higher.

To highlight the effect of true TOs on the responsiveness of BTCP using BLE, we assume, in this set of simulations, that the IP access networks are lossy but the OBS network is not. We compare the performance of BTCP using BLE and Reno by injecting several UDP and other TCP flows in the two IP access networks.

Table V shows the performance differences in terms of their throughput ratio R_t and loss rate ratio R_l as a function of assembly time T_b. Generally speaking, wrong FTO detection in IP access networks can increase the loss rate in the IP access networks. Although it can also increase the throughput of a single BTCP flow, the performance of the other TCP flows' will be degraded due to increased loss rate.

From Table V, we can see that the increase in the loss rate of the network is not significant, and becomes less as the assembly time increases. This is mainly due to the fact that serious congestion that causes a true timeout event can last for some time (considering the long range dependent nature of Internet traffic [5], [8]). Accordingly, even though a BTCP using BLE may treat the first true TO as a FTO, (and transmits data with half of previous congestion window), most likely it will receive another true TO, and eventually its window size will be reduced to less than 3, which will trigger a true TO. Having a large assembly time may result in both poor throughput and loss rate as more true TOs will be mistakenly considered as FTOs by BTCP using BLE.

VI. SUMMARY

This paper have evaluated the performances of three most popular TCP implementations: Reno, New-Reno and SACK in OBS networks. It has been demonstrated that although SACK performs better than Reno and New-Reno, none of them can effectively handle a false timeout (FTO) in a medium or fast TCP flow, which could be quite common in OBS networks. We have proposed a new TCP implementation called Burst TCP or BTCP using three FTO detection methods based on burst length estimation, burst ACK and burst NAK, respectively, which involve the tradeoffs between FTO detection accuracy (and throughput performance), performance and implementation complexity. We have compared BTCP with the current TCP implementations, and shown that BTCP can improve TCP throughput without undermining the congestion control ability of normal TCP.

References

[1] X. Cao, J. Li, Y. Chen, and C. Qiao, “Assembling TCP/IP Packets in Optical Burst Switched Networks”, *Proceeding of IEEE Globecom*, 2002

[2] A. Detti, M. Listanti, “Impact of Segments Aggregation of TCP Reno Flows in Optical Burst Switching Networks”, *IEEE INFOCOM*, vol.3, pp.1803-1812, 2002.

[3] K. Fall and S. Floyd, “Simulation-based Comparisons of Tahoe, Reno and SACK TCP”, *IEEE/ACM Sigcomm*,1996.

[4] S. Gowda, R. Shenai, K. Sivalingam and H. C. Cankaya, , In “Performance Evaluation of TCP over Optical Burst-Switched (OBS) WDM Networks”, *Proceedings of ICC*, 2003.

[5] W.E.Leland, M.S.Taqqu, W.Willinger and D.V.Wilson, “On the self-similar nature of ethernet traffic,”, *IEEE/ACM Transaction on Networking*, vol.2, pp.1-15 1994.

[6] M. Mathis and J. Mahdavi, “Forward Acknowledgement: Refining TCP Congestion Control”, *IEEE/ACM Sigcomm*,1996.

[7] J. Padhye, V. Firoiu, D. Towsley, J. Kurose, “Modeling TCP Throughput: A Simple Model and its Empirical Validation”, *IEEE/ACM Sigcomm*, 1998.

[8] V.Paxson and S.Floyd, “Wide-area traffic: The failure of poisson modeling”, *IEEE/ACM Transaction on Networking*, vol.3, pp.226-224, 1995.

[9] C. Qiao and M. Yoo, “Optical Burst Switching (OBS) - a New Paradigm for an Optical Internet”, *Journal of High Speed Networks*, No.8, pp.69-84, 1999.

[10] J. Ramamirtham and J. Turner, “Time Sliced Optical Burst Switching”, *Proceeding of INFOCOM*, 2003.

[11] W. Stevens, “TCP/IP Illustrated, Volume 1 – The protocols”, 1994.

[12] X. Yu, C. Qiao, Y. Liu and D. Towsley “Performance Evaluations of TCP Traffic Transmitted over OBS Networks”, *Tech. Report 2003-13, CSE Department, SUNY Buffalo*, 2003.