Source-Ordering for Improved TCP Performance over Load-Balanced Optical Burst-Switched (OBS) Networks

Bharat Komatireddy and Vinod M. Vokkarane
Department of Computer and Information Science, University of Massachusetts, Dartmouth, MA 02747, USA
E-mail: {g_bkomatired,vvokkarane}@umassd.edu

***Abstract*— Recent advances in optical switching technology allows for the creation of networks in which data bursts are switched optically at each node, offering a greater degree of flexibility suitable for handling bursty Internet traffic. TCP-based applications account for a majority of data traffic in the Internet; thus understanding and improving the performance of TCP implementations over OBS networks is critical. Previously, several articles show that load-balanced routing improves loss-performance in OBS. In this paper, we identify the ill-effects of load-balanced OBS on TCP performance caused due to false time-outs and false fast-retransmit. We propose *source-ordering* mechanism that significantly improves TCP throughput. We evaluate the performance of the proposed mechanism over different TCP flavors, such as TCP Tahoe, TCP Reno, TCP SACK, and TCP Vegas over a load-balanced OBS network.**

Keywords: Transmission Control Protocol (TCP) and Optical Burst Switching (OBS).

I. INTRODUCTION

Next-generation high-speed optical Internet will be required to support a broad range of emerging applications which may not only require significant bandwidth, but may also have strict requirements with respect to end-to-end delays and reliability of transmitted data.

In optical burst switching (OBS), data to be transmitted is assembled in to bursts and are switched through the network all optically [1]. Each burst has an associated control packet called the burst header packet (BHP) and the BHP is sent ahead of time in order to configure the switches along the bursts' route. In OBS networks, apart from the data channels, each link has one or more control channels to transmit BHPs. BHPs carries information about the burst such as source, destination, burst duration, and offset time. Offset time is the time at which the burst and BHP are separated at the source and the subsequent intermediate nodes. The offset time allows for the BHP to be processed at each intermediate node before the data burst arrives. As the BHP travels from source to destination, it is processed at each intermediate node in order to configure the optical switches accordingly. Then the data burst cuts through the optical layer avoiding any further delays. Bandwidth is reserved only for the duration of the burst, this reservation technique is called just-enough-time (JET) [2].

The primary issue in the OBS core network is contention resolution, since the core does not have any buffers. Contention occurs when two or more bursts contend for the same output port at the same time. There are several contention resolution techniques, such as optical buffering [3], wavelength conversion [4], [5], and deflection routing [6]. These contention resolution techniques are reactive in nature, that try to resolve the contention when it occurs. These contention resolution techniques attempt to minimize the loss based on the local information at the node. An alternative to contention resolution is to avoid contention before it happens.

There is a tremendous need to support reliable connection-oriented end-to-end transport service for supporting new applications, such as the Grid systems. In the recent years, transmission control protocol (TCP)-based applications, such as Web (HTTP), Email (SMTP), peer-to-peer file sharing [7], [8], and grid computing [9], account for a majority of data traffic in the Internet; thus understanding and improving the performance of TCP implementations over OBS networks is critical. The important TCP flavors are TCP Tahoe [10], TCP Reno [11], [12], [13], TCP SACK [14], and TCP Vegas [15], [16], [17]. The fundamental assumption of all these TCP flavors is that the underlying medium is electronic in nature, and that the packets experience queueing (buffering) delays during congestion in the electronic IP routers along the path of the TCP flow. Over the years, TCP has undergone significant changes in terms of developing new congestion control mechanisms and handling issues concerning the need for high-bandwidth at the presence of long end-to-end delays between the senders and the receivers [18].

TCP flavors primarily differ in their implementation of congestion control mechanisms. TCP and its various flavors can be classified into three categories based on congestion control mechanisms, they are loss-based, delay-based, and rate-based. TCP Tahoe, TCP Reno and TCP SACK are loss-based congestion-control techniques

that use packet losses to estimate the available bandwidth in networks. TCP Tahoe is one of the first and simplest loss-based congestion control versions. TCP Reno and TCP SACK are widely deployed TCP versions in the Internet. TCP Reno and TCP SACK employs loss-based congestion-control using *time-out* (TO) and *fast-retransmit* (FR) based mechanisms [19]; while TCP Tahoe employs only *timeout* (TO) based congestion control.

On the other hand, delay-based TCP flavors, such as TCP Vegas [16], use delay measurements to estimate available bandwidth in the network. The queueing delay measured in TCP can provide information about the degree of network congestion, which will make TCP implementation easier to stabilize a network with a target fairness and high utilization. The performance of TCP Vegas has been evaluated in [16], [20]. The paper suggests that TCP Vegas achieves 30% to 70% higher throughput than TCP Reno by reducing the number of packet retransmissions.

Recently, a third kind of TCP congestion control mechanism, rate-based congestion control has been proposed. A rate-based eXplicit Control Protocol (XCP) has been proposed in [21], where available network bandwidth is estimated based on the explicit feedbacks from routers in the networks. In [22], Rate Control Protocol (RCP) is proposed, which is similar to XCP except that the router assigns a single rate to all flows that pass through it.

Due to the bufferless nature of OBS core network and the one-way based signaling scheme, the OBS network will suffer from random burst losses even at low traffic loads. One problem that arises when TCP traffic traverses over OBS networks is that the random burst loss may be falsely interpreted as network congestion by the TCP layer. For example, if a burst that contains all of the segments of a TCP sending window is dropped due to contention at a low traffic load, then the TCP sender times out, leading to false congestion detection. This false congestion detection is referred to as a *false time-out* (FTO) [23]. When the TCP sender detects this (false) congestion, it will trigger the *slow start* congestion control mechanism, which will result in the TCP throughput being reduced. Another example is when a random burst loss triggers TCP fast retransmission for the case in which segments in a TCP sending window are assembled into multiple bursts. The burst loss will be interpreted as light network congestion and will trigger one or more TCP-layer fast retransmissions. Recently, few works have evaluated TCP throughput over a OBS network [24], [25], [26]. However, these works assume a constant random burst loss probability in the OBS network, and do not take into account TCP false congestion detection.

Load-balanced routing is an approach to implement contention avoidance in OBS [27]. Load-balanced routing involves two stages, *route calculation* and *route selection*. Both route calculation and route selection can be implemented in a static or a dynamic manner. In this paper, we adopt the a load-balanced routing technique with static route-calculation and dynamic route-selection as proposed in [27]. At every τ seconds, all the ingress OBS node dynamically selects the least-congested path (among the two static link-disjoint minimum-hop paths) to all their destination nodes using the cumulative congestion-information of all the link along the two pre-calculated paths. A link is said to be congested, if offered load on Link (i, j), $L_{i,j} \geq P_{max}$, where P_{max} is the maximum load threshold on a link. Let τ_s and τ_d be the duration of successful burst arrivals and dropped burst arrivals during the interval τ, respectively. The offered load on each of the node's outgoing link is expressed as the duration of all arriving bursts over the interval τ, is given by, $L_{i,j} = \frac{\tau_s+\tau_d}{\tau}$.

In this paper, we propose a *source ordering* mechanism to minimize the number of *false time-outs* (FTOs) and *false fast retransmit* (FFR). FTOs and FFRs are essentially false congestion indicators at the OBS-layer that are perceived as true TOs or FRs by the TCP-layer. In source ordering, we aim to neutralize the negative impact of the delay-differential between multiple transmission paths in the OBS network on higher-layer TCP performance. The remainder of the paper is organized as follows. Section II discusses the issue of supporting TCP over an independently load-balanced OBS network. Section III describes the proposed *source ordering* mechanism in order to improve TCP performance over an OBS network. provides background information on congestion-based load-balanced routing in OBS networks. Section IV discusses the simulations results and Section V concludes the paper.

II. TCP OVER LOAD-BALANCED OBS

Loss-based TCP congestion-control mechanisms generally include slow start, congestion avoidance, fast retransmission, and fast recovery.

In TCP Tahoe, a TCP segment loss is detected by a *time-out* (TO). A TO loss is detected by a *retransmission time-out* (RTO), when an acknowledgment for a segment is not received within a certain period of time. TCP interprets a TO loss as a loss due to network congestion; hence, the TCP sender retransmits the lost segment and enters into a *slow start* phase.

In TCP Reno, if a TCP segment is lost, there are two types of loss indications: *time-out* (TO) and *fast*

retransmit (FR). TCP Reno interprets a TO loss as a loss due to heavy network congestion; hence, the TCP sender retransmits the lost segment and enters into a *slow start* phase. A fast retransmission is triggered when a TCP sender receives three duplicate ACKs, which indicates that a packet is lost due to light network congestion; hence, the TCP sender enters into *fast retransmission* and *fast recovery* without waiting for RTO.

In TCP Reno, after receiving triple duplicate ACKs, the source retransmits one lost segment, reduces the size of congestion window by half, and enters into a *fast recovery* phase. During the fast recovery phase, the source increases the congestion window by one segment for each duplicate ACK that it receives. After receiving half a window of duplicate ACKs, the congestion window size will be the same as the window size prior to the fast retransmit phase. Thus, the source can send a new packet for each additional duplicate ACK that it receives. The source exits fast recovery upon the receipt of the ACK that acknowledges the retransmitted lost segment, and enters into a *congestion avoidance* phase. TCP Reno is suitable for single segment loss in the sending window, but does not handle multiple losses well.

TCP SACK was proposed as an enhancement to TCP Reno. TCP Reno uses an acknowledgement number field that contains a cumulative acknowledgement, indicating the TCP receiver has received all the data up to the indicated byte. A *selective acknowledgement option* allows receivers to additionally report non-sequential data they have received. In the load-balanced routing protocol packets can be reached out-of-order because of the delay difference between the primary path and secondary path. TCP SACK should perform better than TCP Reno over a load-balanced OBS due to selective acknowledgement.

In the absence of TCP SACK, TCP Reno has performance problems when multiple packets are dropped from one window of data. These problems result from the need to await retransmission timer expiration before re-initiating the data flow. TCP SACK is based on a conservative extension of the TCP Reno congestion-control algorithms with the addition of selective acknowledgements and selective retransmission. With TCP SACK a sender has a better idea of exactly which packets have been successfully delivered. Given such information a sender can avoid unnecessary delays and retransmissions, resulting in improved throughput.

All the above loss-based approach may cause *false time out* (FTO) and *false fast retransmit* (FFR) in an OBS network when there is no serious congestion in the network. FTO is caused because of random burst contention instead of IP router buffer overflow. While FFR is caused because of the out-of-order arrival of TCP segments at the destination due to the delay-differential between the primary and the alternate paths of a load-balanced OBS network.

TCP Vegas is a delay-based congestion-control mechanism. TCP Vegas adopts a more sophisticated bandwidth estimation scheme. It uses the difference between expected and actual flows rates to estimate the available bandwidth in the network [28]. TCP Vegas retransmission mechanisms use a fine-grained timer for loss detection, and treats the receipt of certain ACKs as a trigger to check for time-outs. Whenever a loss is detected via a triple-duplicate ACK, Vegas reduces the window by a quarter instead of half.

TCP Vegas extends the TCP Reno's retransmission mechanism as follows [15]: First, TCP Vegas records the system clock each time a segment is sent. When an ACK arrives the sender reads the clock again and calculates the RTT for the relevant segment. The sender uses this accurate RTT estimate to decide to retransmit in the following two situations:

1. When a duplicate ACK is received, sender checks to see if the difference between the current time and the $timestamp$ recorded for the relevant segment is greater than the timeout value. If it is, then TCP Vegas retransmits the segment without having to wait for three duplicate ACKs. In many cases, losses are either so great or the window so small that the sender will never receive three duplicate ACKs, and therefore, Reno would have to rely on the coarse-grain timeout mentioned above.
2. When a non-duplicate ACK is received, if it is the first or second one after a retransmission, TCP Vegas again checks to see if the time interval since the segment was sent is larger than the timeout value. If it is, then TCP Vegas retransmits the segment. This will catch any other segment that may have been lost previous to the retransmission without having to wait for a duplicate ACK.

Static load-balanced routing techniques uses two path to transmit data between each source-destination pair, a primary path and an alternate path. The alternate being longer than and link-disjoint from the primary. In such a scenario, the bursts transmitted on the alternate path incurs longer delay compared to the bursts transmitted on the primary path. The path delay-differential (δ) encountered may cause out-of-order reception of TCP segments (IP packets) at the destination, resulting in FTOs and FFRs.

Consider the following illustration scenario to better understand the issue of FTOs and FFRs due to load-balanced routing in OBS networks. In Fig. 1(a), Burst B1 consisting of three segments [S1,S2,S3] is transmitted

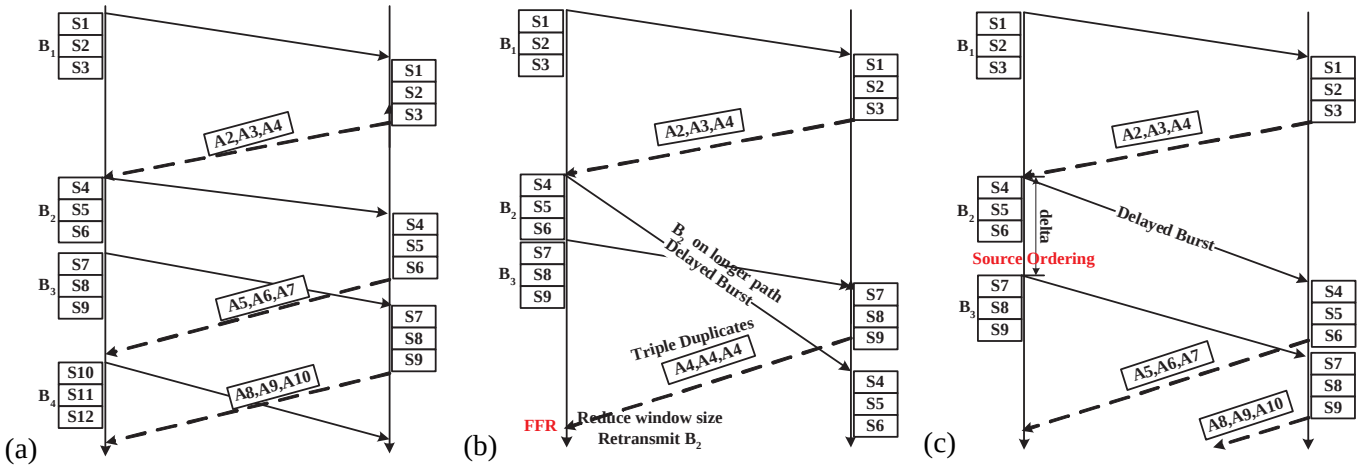


Fig. 1. (c) TCP-over-OBS with fixed-routing, (b) FFR in TCP over load-balanced OBS, and (c) source-ordering to minimize FFR (and FTO) in TCP over load-balanced OBS.

and the corresponding acknowledgements [A2,A3,A4] are received. Assuming that the flow is in slow-start phase, congestion window doubles and the sender can possibly send at least six packets. Burst B2 consisting of segments [S4,S5,S6] is sent followed by Burst B3 consisting of segments [S7,S8,S9] and so on. In Fig. 1(b), load-balanced routing in the OBS-layer may result in Burst B2 and Burst B3 being transmitted on two different paths, say B2 on secondary path and B3 on the primary shortest path. The Burst B2 [S4,S5,S6] gets delayed due to longer alternate path, Burst B3 [S7,S8,S9] reaches destination before Burst B2 since Burst B3 contains three out-of-order segments [S7,S8,S9], the receiver will send three duplicate ACKs [A4,A4,A4]. This results in FFRs at the TCP sender. Note that if the path delay-differential is significant, TCP sender may experience FTOs.

III. Source Ordering

In order to neutralize the negative impact of the path delay-differential caused by load-balanced routing in OBS, we propose *source ordering*. In OBS, all the ingress edge nodes implement source routing to transmit burst to the corresponding destinations. In source ordering, the ingress edge node pre-calculates the path delay-differential between the primary minimum-hop path and the alternate link-disjoint second minimum-hop path, $\delta = \mid P_1 - P_2 \mid$, where P_1 is the end-to-end delay on the primary path and P_2 is the end-to-end delay on the alternate path.

We observe that every time the ingress node performs a path-switch from the longer alternate path to the shorter primary path, some of the bursts transmitted on the primary path may overtake the previously transmitted bursts on the longer alternate path before reaching the destination. Every time we perform a long-to-short path-switch, this scenario is quite common especially when the δ value is large. This differential in path-delay can result in FFRs and possibly FTOs (refer Fig. 1(b)). In source ordering, every time a long-to-short path-switch occurs, we electronically buffer the bursts for δ seconds before we start transmitting on the shorter path.

In Fig. 1(c), every time a long-to-short path-switch occurs we delay the burst for the amount of time equivalent to the path delay-differential of the two paths, using electronic buffering at the ingress OBS node. Note that the ingress node is aware of the path delay-differential since OBS implements source-routing.

IV. Simulation Results

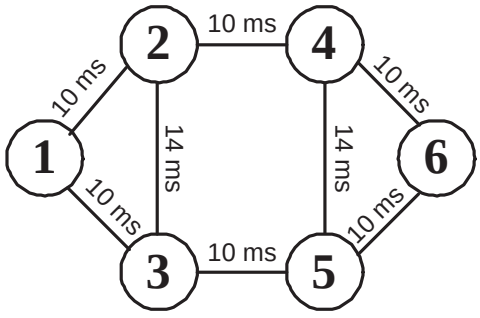


Fig. 2. Network Topology.

We develop a NS-2 simulation to evaluate the performance of TCP Tahoe, TCP Reno, TCP SACK, and TCP Vegas over load-balanced OBS. The simulation network used is a 6-node OBS network depicted in Fig. 2. Every fiber link has four data channels operating at a transmission rate of 1 Gb/s. Timer-based burst assembly algorithm [29], [30] is adopted, with a

timer value of 20 ms. The OBS network implements JET-based signaling with LAUC-VF channel scheduling algorithm [31]. The offset time is 50 μs and the per-node burst header processing time is 5 μs. We set up 100 TCP flows from Node 1 to Node 6 using 100 FTP sources generating packets with an average size of 2 KB. Load-balanced routing is implemented in the OBS-layer with interval, τ = 1 ms and congestion threshold, $\rho_{max} = 0.5$ [27]. We can observe from the network topology (refer Fig. 2) that the primary shortest-hop path is $P1(N1 - N2 - N4 - N6)$ and the secondary link-disjoint path is $P2(N1 - N3 - N5 - N6)$. In order to simulate impact of path delay-differential between primary and secondary paths on TCP performance, we vary propagational delay of Link $L_{3,5}$ from 10 ms to 110 ms to generate δ values ranging from 0 ms to 100 ms. In order to compare TCP performance with and without load-balanced OBS routing, we plot for TCP over OBS without load-balanced routing (referred as $baseline$). In addition to TCP traffic from Node N1 to Node N6, all the other nodes have a 2 Mb/s constant-bit-rate UDP traffic between them. We evaluate the following performance metrics for the different TCP version:

- *On-line TCP throughput (in bytes/second)*: number of bytes received per second at the TCP sink.
- *Cumulative average TCP throughput (in bytes/second)*: cumulative average of number of bytes received per second at the TCP sink.
- *Congestion window size (in bytes)*: value of congestion window in bytes at the end of every second.

A. TCP Tahoe over Load-Balanced OBS

Figure 3(a) plots on-line TCP Tahoe throughput for all the 100 TCP flows from N1 to N6 over a simulation period of 50 seconds. In the graph, we can observe that on-line throughput when $\delta = 0$ is significantly higher than on-line throughput when $\delta = 100$. On-line throughput when $\delta = 0$ is similar to that of $baseline$ (without load-balanced OBS), this is due to the fact that TCP Tahoe uses time-outs as the only packet loss indicator. A slight out-ordering of TCP segments at the receiver does not trigger fast retransmission.

Figure 3(b) plots cumulative average TCP throughput for all the 100 TCP flows from N1 to N6 over a simulation period of 50 seconds. In the graph, we can observe that average throughput when $\delta = 0$ is significantly higher than the average throughput when $\delta = 100$. At certain points on the graph, average throughput when $\delta = 0$ is higher than the $baseline$ case. In TCP Tahoe as retransmission is done only when the time-out occurs, there is no issue of FFRs. In summary, there is marginal benefit to implement load-balanced OBS (even with source ordering) versus implementing OBS without load-balanced routing for TCP Tahoe-based flows. We have also observed that the TCP throughput further drops with increase in $delta$ values.

B. TCP Reno over Load-Balanced OBS

Figure 4(a) plots on-line throughput for all the 100 TCP flows from N1 to N6 over a simulation period of 50 seconds. We can observe from the graph that on-line throughput when the propagational delay of both the primary and secondary paths are identical ($\delta = 0$) is significantly higher than the online throughput of the network, where the delay difference is equal to 100 ($\delta = 100$). This difference in performance is due to the fact that TCP Reno uses both triple-duplicates and time-outs as packet loss indicators. TCP performance with a $\delta = 100$ will results in several FTOs and FFRs leading to lower throughput. We can also observe TCP throughput over load-balanced OBS (when δ=0) outperforms the $baseline$ scenario (without load-balancing).

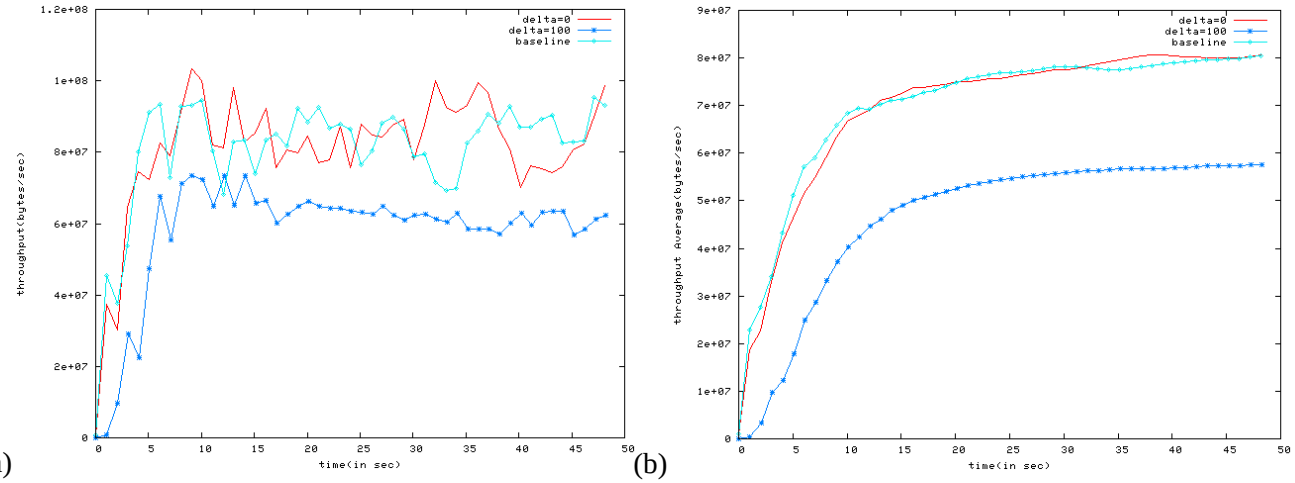
Figure 4(b) plots the cumulative average throughput for all the 100 TCP flows from N1 to N6 over a simulation period of 50 seconds. We again observe that $\delta = 0$ has higher average throughput than $\delta = 100$ and $baseline$. This is because of the FTOs and FFRs that we have explained before.

Figure 4(c) plots the congestion window size versus simulation time for a single flow (Flow 1) out of the 100 TCP flows from N1 to N6 over a simulation period of 50 seconds. We can observe from the graph that the congestion window size larger and grows faster for $\delta = 0$ compared to $\delta = 100$ and $baseline$.

Fig 4(d) supports our claim by plotting number the cumulative FRs versus time for the different scenarios. We observe that there are no FRs for the case of $\delta = 0$ ms. The number of FRs graph and congestion window graph justify the reason for decrease in throughput when difference in delay between the primary path and secondary path is greater.

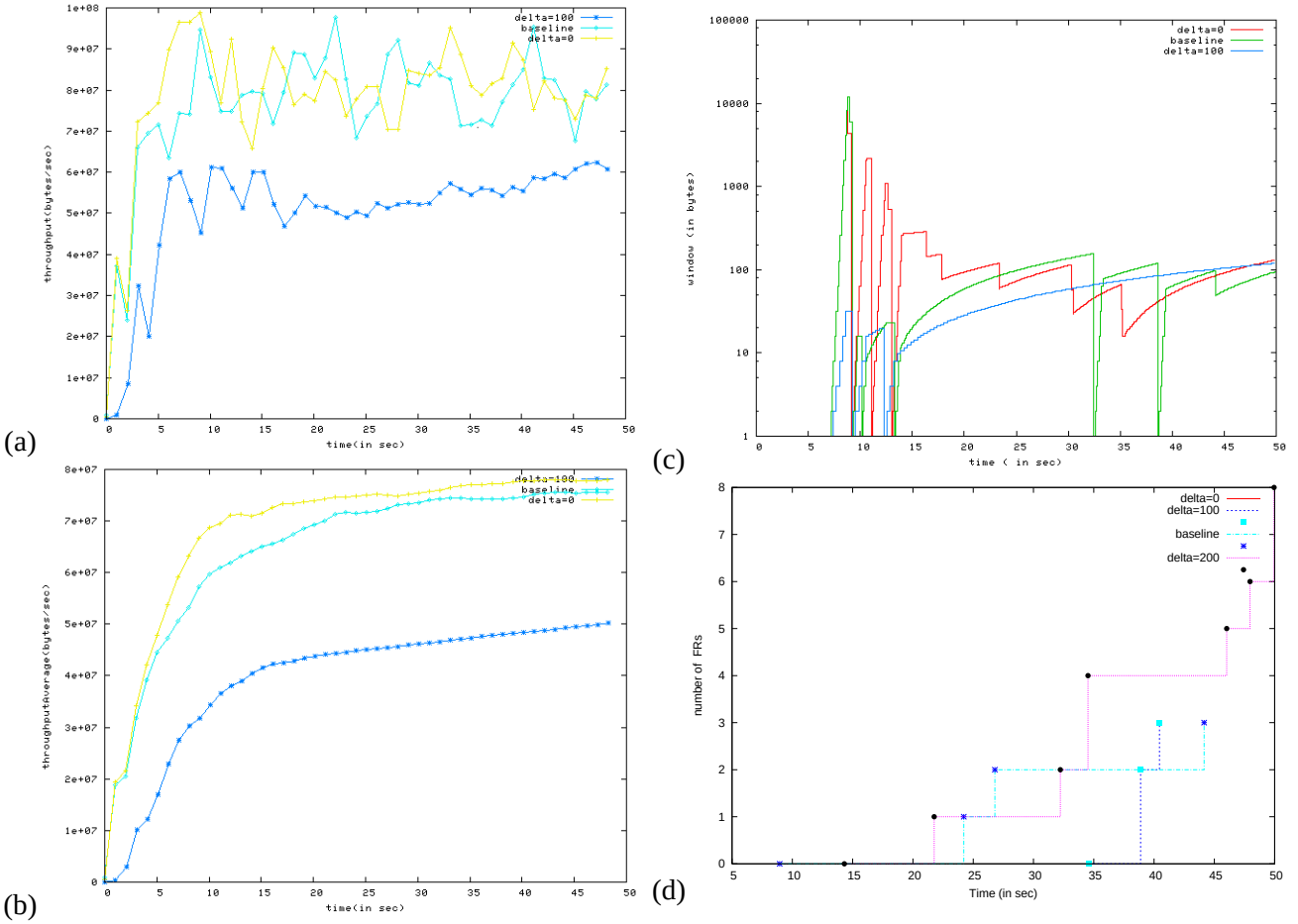
C. TCP SACK over Load-Balanced OBS

Figure 5(a) plots on-line throughput for all the flows (100) from $N1$ to $N6$ over a simulation period of 50 seconds for all the TCP traffic. In the graph, we can observe that throughput is higher when $\delta = 0$ compared to when $\delta = 100$ due to FTOs and FFRs as explained before. We can also observe TCP throughput over load-balanced OBS (when δ=0) outperforms the $baseline$ scenario (without load-balancing). Fig. 5(b) plots the cumulative average throughput for all 100 TCP



(a) (b)

Fig. 3. TCP Tahoe: (a) On-line TCP throughput vs. simulation time. (b) Cumulative average throughput vs. simulation time.



(a) (c) (b) (d)

Fig. 4. TCP Reno: (a) On-line TCP throughput vs. simulation time. (b) Cumulative average TCP throughput vs. simulation time. (c) Congestion window size vs. simulation time. (d) Number of cumulative fast-retransmits (FRs) vs. simulation time.

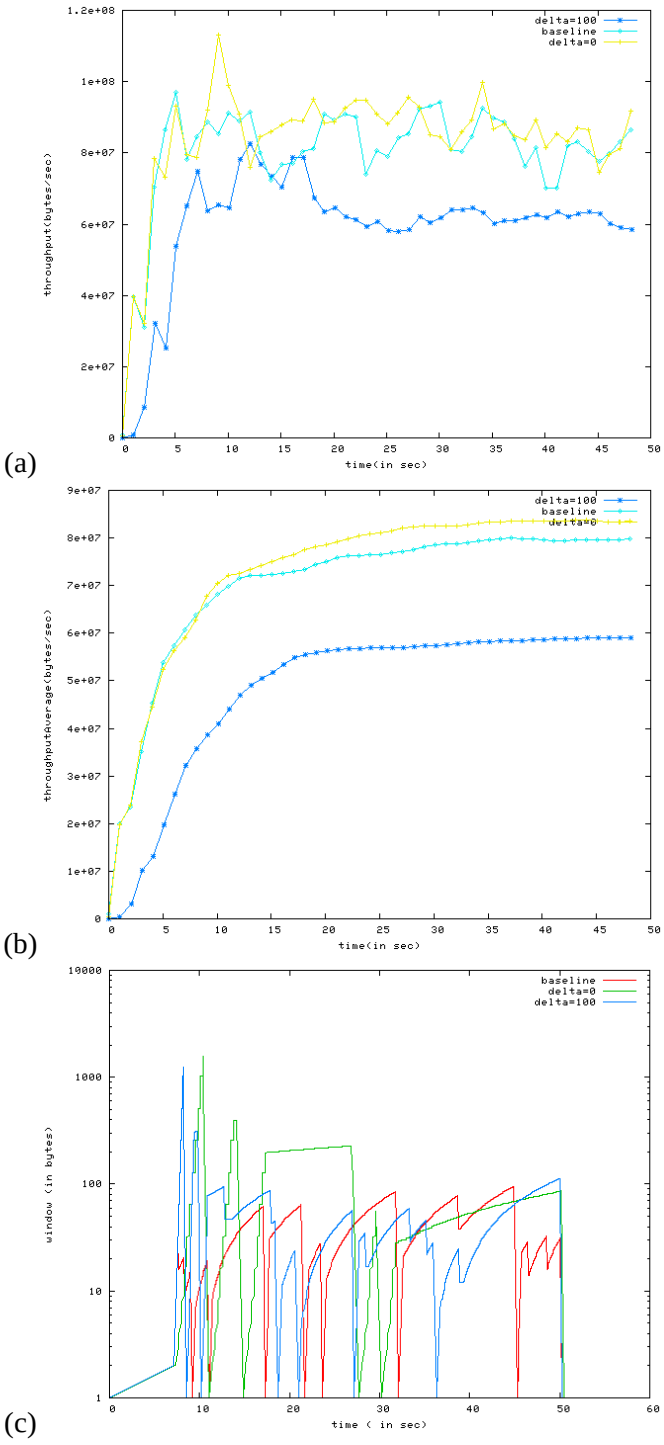


Fig. 5. TCP SACK: (a) On-line TCP throughput vs simulation time. (b) Average TCP throughput vs simulation time. (c) Congestion window size versus simulation time.

flows from $N1$ to $N6$ over a simulation period of 50 seconds. We again observe that $\delta = 0$ has higher average throughput than $\delta = 100$ and $baseline$ due to FTOs and FFRs.

Figure 5(c) plots the congestion window size versus simulation time for a single flow (Flow 1) out of the 100 TCP flows from N1 to N6 over a simulation period of 50 seconds. We can observe from the graph that the congestion window size larger and grows faster for $\delta = 0$ compared to $\delta = 100$ and $baseline$. We can also observe that throughput of TCP SACK is better than both TCP Tahoe (Fig 3) and TCP Reno (Fig 4).

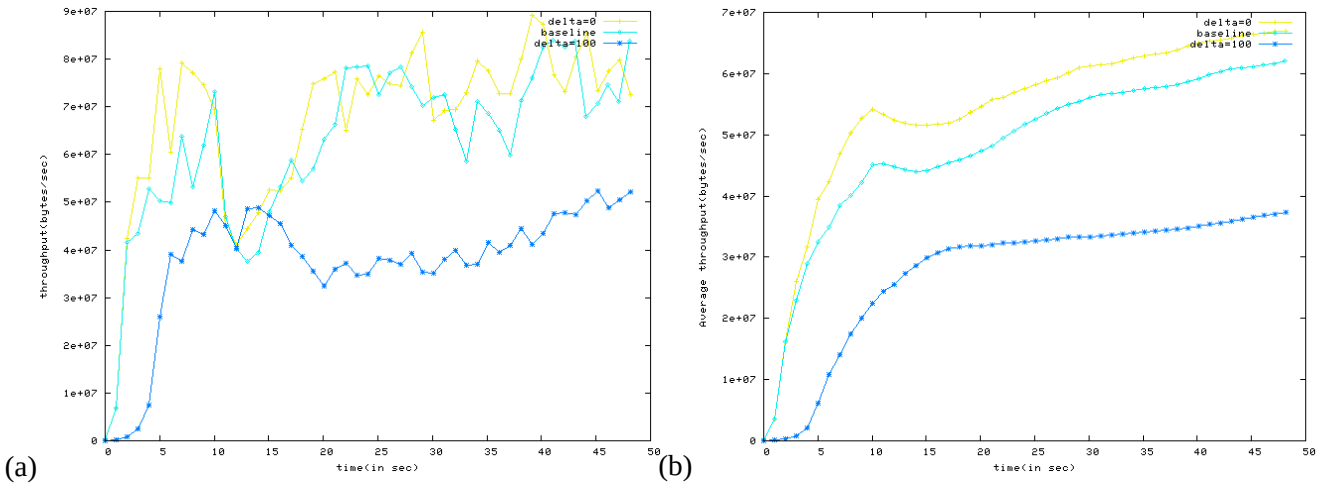
D. TCP Vegas over Load-Balanced OBS

As mentioned before, when a duplicate ACK is received, sender checks to see if the difference between the current time and the $timestamp$ recorded for the relevant segment is greater than the timeout value. If so, TCP Vegas retransmits the segment without having to wait for three duplicate ACKs. Due to this enhancement a pair of routes with delay-differential in a load-balanced OBS can cause more harm to TCP Vegas flows then TCP Reno/SACK flows due to out-of-order delivery of packet. Figure 6(a) plots on-line throughput for all the 100 TCP Vegas flows from $N1$ to $N6$ over a simulation period of 50 seconds. In the graph, we can observe that throughput is higher when $\delta = 0$ compared to when $\delta = 100$ due to FTOs and FFRs as explained before. We can also observe TCP throughput over load-balanced OBS (when δ=0) outperforms the $baseline$ scenario (without load-balancing).

Figure 6(b) plots the cumulative average throughput for all 100 TCP flows from $N1$ to $N6$ over a simulation period of 50 seconds. We again observe that $\delta = 0$ has higher average throughput than $\delta = 100$ and $baseline$ due to FTOs and FFRs.

V. Conclusion

In this paper, we have evaluated the performance of different TCP flavors over a load-balanced OBS. In load-balanced routing, two routes are first calculated statically and the least-congested route is selected dynamically for data transmission. We identify the ill-effects of OBS-layer load-balanced routing on higher-layer TCP performance. Through extensive simulations it is clear that the value of the path delay-differential has a significant impact on the higher-layer TCP performance. We propose a simple source-ordering approach that maintains the order of the bursts using electronic buffers at the ingress OBS edge node, so as to minimize the number of false time-outs and false fast-retransmit. We observe



(a) (b)

Fig. 6. TCP Vegas: (a) On-line throughput vs. simulation time. (b) Cumulative average throughput vs. simulation time.

that source-ordering can improve the TCP throughput by up to 400%.

An important area of future work is to implement load-balanced routing with ReorderingRobust (RR-TCP) [32] in order to avoid false fast retransmits and false time-outs. Another area of future work is to implement TCP over OBS with burst segmentation [33]. Burst segmentation will increase the probability of a burst reaching the destination, leading to reduction of false time-outs (and false fast-retransmits). This can also have a significant positive impact on the TCP-over-OBS performance.

VI. Acknowledgements

This work was supported in part by NSF Grant CNS-0626798.

References

[1] J.P. Jue and V.M. Vokkarane, *Optical Burst Switched Networks*, Springer, 2005.

[2] C. Qiao and M. Yoo, "Optical burst switching (OBS) - a new paradigm for an optical Internet," *Journal of High Speed Networks*, vol. 8, no. 1, pp. 69–84, January 1999.

[3] I. Chlamtac, A. Fumagalli, L. G. Kazovsky, and et al., "CORD: Contention resolution by delay lines," *IEEE Journal on Selected Areas in Communications*, vol. 14, no. 5, pp. 1014–1029, June 1996.

[4] B. Ramamurthy and B. Mukherjee, "Wavelength conversion in WDM networking," *IEEE Journal on Selected Areas in Communications*, vol. 16, no. 7, pp. 1061–1073, September 1998.

[5] A. Bononi, G. A. Castanon, and O. K. Tonguz, "Analysis of hot-potato optical networks with wavelength conversion," *IEEE/OSA Journal of Lightwave Technology*, vol. 17, no. 4, pp. 525–534, April 1999.

[6] S. Yao, B. Mukherjee, S. J. B. Yoo, and S. Dixit, "A unified study of contention-resolution schemes in optical packet-switched networks," in *IEEE/OSA Journal of Lightwave Technology*, March 2003.

[7] I. Stoica and et. al., "Chord: A scalable peer-to-peer lookup protocol for internet applications," in *Proceedings of ACM SIGCOMM*, 2001.

[8] K. Gummadi, R. Dunn, S. Saroiu, S. Gribble, H. Levy, and J. Zahorjan, "Measurement, modeling, and analysis of a peer-to-peer file-sharing workload," in *Proceeding of ACM SIGMETRICS*, 2003.

[9] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the Grid: Enabling scalable virtual organizations," *Intenational Journal of High Performance Computing Applications*, vol. 15, pp. 200–222, 2004.

[10] K. Fall and S. Floyd, "Simulation-based comparisons of tahoe, reno and sack tcp," *ACM SIGCOMM Computer Communication Review*, vol. 26, pp. 5–21, July 1996.

[11] V. Jacobson, "Congestion avoidance and control," in *Proceedings, ACM SIGCOMM*, 1988.

[12] W. Stevens, "TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms," *RFC 2001*, 1997.

[13] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP Reno performance: A simple model and its empirical validation," *IEEE/ACM Transactions on Networking*, vol. 8, no. 2, April 2000.

[14] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP selective acknowledgement options," *RFC 2018*, 1996.

[15] L. Brakmo, S. O'Malley, and L. Peterson, "TCP Vegas: New techniques for congestion detection and avoidance," in *Proceedings, SIGCOMM,*, August 1994, pp. 24–35.

[16] L. Brakmo and L. Peterson, "TCP Vegas: End to end congestion avoidance on a global internet," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 8, pp. 1465–1480, October 1995.

[17] E. Weigle and W.-C. Feng, "A case for TCP Vegas in high-performance computational grids," in *Proceedings, 10th IEEE International Symposium on High Performance Distributed Computing (HPDC)*, August 2001.

[18] W. Feng and P. Tinnakornsrisuphap, "The failure of TCP in high-performance computational grids," in *Proceedings, Supercomputing Conference*, 2000.

[19] V. Jacobson, "Congestion avoidance and control," in *ACM SIGCOMM*, 1989.

[20] J. Ahn, P. Danzig, Z. Liu, and L. Yan, "Evaluation of TCP Vegas: emulation and experiment," *Computer Communication Review*, vol. 25, pp. 185–95, October 1995.

[21] D. Katabi, M. Handley, and C. Rohrs, "Congestion control for high bandwidth-delay product networks," in *Proceedings, ACM SIGCOMM*, 2002.

[22] N. Dukkipati, M. Kobayashi, R. Zhang-Shen, and N. McKeown, "Processor sharing flows in the internet," in *Thirteenth*

International Workshop on Quality of Service (IWQoS), June 2005.

[23] X. Yu, C. Qiao, and Y. Liu, "TCP implementations and false time out detection in OBS networks," in *Proceedings, IEEE Infocom*, March 2004.

[24] A. Detti and M. Listanti, "Impact of segments aggregation on TCP Reno flows in optical burst switching networks," in *Proceedings, IEEE Infocom*, 2002.

[25] X. Yu, C. Qiao, Y. Liu, and D. Towsley, "Performance evaluation of TCP implementations in OBS networks," in Technique Report 2003-13, *The State University of New York at Buffalo*, 2003.

[26] S. Gowda, R. Shenai, K. Sivalingam, and H. C. Cankaya, "Performance evaluation of TCP over optical burst-switched (OBS) WDM networks," in *Proceedings, IEEE International Conference on Communications (ICC)*, May 2003, vol. 2, pp. 1433–1437.

[27] G.P.V. Thodime, V. M. Vokkarane, and J. P. Jue, "Dynamic congestion-based load balanced routing in optical burst-switched networks," in *Proceedings, IEEE Globecom*, December 2003, vol. 5, pp. 2694–2698.

[28] J. Mo, R.J. La, V. Anantharam, and J. Walrand, "Analysis and comparison of tcp reno and vegas," in Proceedings, IEEE INFOCOM, 1999.

[29] A. Ge, F Callegati, and L. Tamil, "On optical burst switching and self-similar traffic," *IEEE Communications Letters*, vol. 4, no. 3, March 2000.

[30] V. M. Vokkarane, K. Haridoss, and J. P. Jue, "Threshold-based burst assembly policies for QoS support in optical burst-switched networks," in *Proceedings, SPIE OptiComm*, July 2002, vol. 4874, pp. 125–136.

[31] Y. Xiong, M. Vanderhoute, and H.C. Cankaya, "Control architecture in optical burst-switched WDM networks," *IEEE Journal on Selected Areas in Communications*, vol. 18, no. 10, pp. 1838–1854, October 2000.

[32] M. Zhang, B. Karp, S. Floyd, and L. Peterson, "Rr-tcp: a reordering-robust tcp with dsack," in Proceedings, IEEE ICNP, Nov. 2003, pp. 95–106.

[33] V. M. Vokkarane and J. P. Jue, "Burst segmentation: An approach for reducing packet loss in optical burst switched networks," *SPIE Optical Networks Magazine*, vol. 4, no. 6, pp. 81–89, November-December 2003.