

High-Speed RR-TCP Over Load-Balanced Optical Burst-Switched (OBS) Networks

Neal Charbonneau, Bharat Komatireddy, and Vinod M. Vokkarane

Department of Computer and Information Science, University of Massachusetts, Dartmouth, MA

E-mail: {u_ncharbonne, vvokkarane}@umassd.edu

Abstract—TCP-over-OBS is a promising transport paradigm to support next-generation Internet. Load-balanced routing generally improves loss performance over OBS. We identify that implementing TCP over load-balanced OBS could lead to persistent out-of-order delivery of TCP segments resulting in unnecessary timeouts and fast retransmissions. In this paper we evaluate High-Speed Reordering-Robust TCP (HS-RR-TCP) over OBS networks and compare HS-RR-TCP to an OBS layer approach called source ordering. Through extensive simulations we show that HS-RR-TCP achieves significant improvement in TCP performance. We also find that source ordering provides minimal improvement over the higher layer HS-RR-TCP approach.¹

Keywords: Load-balanced routing, HS-TCP, OBS.

I. INTRODUCTION

Next-generation high-speed optical Internet will be required to support a broad range of emerging applications that 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) that is sent ahead of time in order to configure the switches along the burst's 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 separation time between the burst and its BHP 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 switches avoiding any further delays. Bandwidth is reserved only for the duration of the burst, this reservation technique is called just-enough-time (JET) [2].

In the recent years, TCP-based applications, such as WWW (HTTP), Email (SMTP), peer-to-peer file sharing [3], [4], and grid computing [5], 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 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.

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 and enters slow start, leading to false congestion detection. If the sender's window is instead spread across multiple bursts, a burst drop may lead to fast retransmission.

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, wavelength conversion, deflection routing [6], and burst segmentation [7]. These contention resolution techniques are reactive in nature, they 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.

Load-balanced routing is an approach to implement contention avoidance in OBS [8] [9] [10]. 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 a load-balanced routing with static route-calculation and dynamic route-selection as proposed in [8]. At every τ seconds, all the ingress OBS nodes dynamically select the least-congested path (among two static link-disjoint minimum-hop paths) to all their destination nodes using the cumulative congestion-information of all the links along the two pre-calculated paths. A link is said to be congested if the offered load on link (i, j) , $L_{i,j} \geq \rho_{max}$, where ρ_{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, $L_{i,j} = \frac{\tau_s + \tau_d}{\tau}$. Load-balancing leads to reordering of bursts that degrade TCP's performance as discussed in Section II.

It was shown in [11] that load-balancing degrades TCP's performance, so an OBS layer technique called *source ordering* was introduced to handle the reordering caused by load-balancing. It was shown that TCP's performance with source ordering and load-balancing was better than TCP over a normal OBS network without load-balancing. This shows that by handling the reordering issue, we can take advantage of load-balancing without hurting TCP performance.

Source ordering requires OBS layer modifications for its implementation which will be discussed in Section III, so in this paper we aim resolve the reordering independently at the higher TCP-layer. Reordering-Robust TCP (RR-TCP) has

¹This work was supported in part by the National Science Foundation (NSF) under grant CNS-0626798.

been proposed [12] to handle out-of-order reception of packets at the TCP-layer. In this paper, we evaluate the performance of HS-RR-TCP (RR-TCP with HS-TCP’s modifications [13]) over a load-balanced OBS network under different network conditions and compare it to the source ordering technique.

The remainder of the paper is organized as follows. Section II discusses the issues of TCP over a load-balanced OBS network. Section III discusses source ordering. Section IV describes the *RR-TCP* mechanism, then Section V provides our simulation results. Lastly, Section VI concludes the paper.

II. TCP OVER LOAD-BALANCED OBS

In a load-balanced network, each ingress uses two link-disjoint paths for routing, a primary path and an alternate path. As described in the introduction, when the congestion on one path is greater than the threshold ρ_{max} , the ingress switches to the other path. If the path it switches to has a longer or equal delay, then reordering will not occur. There is a problem, however, if the path it switches to has shorter delay. New bursts sent on the shorter path have the possibility of reaching the destination before bursts sent on the longer path prior to the path switch. This is illustrated in Fig. 1. We can see burst B_2 is transmitted on the longer path while burst B_3 is transmitted on the shorter path due to load-balancing. This will cause duplicate acknowledgements and the TCP sender will try to recover the packets using fast retransmission even though there is no actual loss.

Fast retransmission is a process of retransmitting the lost TCP segment at a faster pace than timeouts. During this process, the TCP sender’s congestion window size is reduced to half. Fast retransmit requires a number of duplicate ACKs to be received at the sender before it concludes that the network has dropped a packet. This parameter is called *dupthresh* and is set to three by default. TCP does not know whether duplicate ACKs are generated by a lost segment or reordering of segments, so TCP sender waits for a small number of duplicate ACKs to be received to confirm packet loss. TCP essentially misinterprets packet reordering caused by load-balanced routing to be packet loss, and triggers *false fast retransmissions*.

Since load-balanced routing reorders packets persistently and packet reordering is interpreted as packet loss, fast retransmission re-sends packets that have not been lost, wasting network bandwidth and keeping window-size unnecessarily small. The goal of HS-RR-TCP over OBS is to allow TCP to adapt to these duplicate acknowledgements and prevent false loss detection. In this paper we use HS-TCP [13] with SACK and RR-TCP to better utilize the high speed OBS network.

III. RELATED WORK: SOURCE ORDERING

Source ordering [11] is an OBS layer mechanism to eliminate the impact of the path delay-differential caused by load-balanced routing. With source ordering, each ingress calculates the delay differential, δ , between its primary path and its alternate path.

When an ingress node changes paths from the longer alternate path to the shorter primary path, the new traffic placed on the shorter path may reach the destination before the traffic on the longer path. As we will see later, the delay differential does not need to be large for this to happen. As a result of the traffic on the shorter path arriving first, TCP will experience false fast retransmissions.

Source ordering eliminates this by using electronic buffers to store bursts whenever a path switch from the longer path to the shorter path occurs. The bursts are buffered for δ time units. The ingress is able to do this because OBS implements source-routing.

Source ordering is illustrated in Fig. 2. The figure shows that reordering is prevented by buffering the burst destined for the short path by δ time units. Comparing this to Fig. 1 we can see that it is able to prevent false fast retransmissions.

While source ordering works well, it requires the use of electronic buffers at the ingress nodes to store incoming data for the length of the delay differential. At high rates this could lead to buffers in the 100s of MBs and separate buffers are required for each source-destination pair. Another implementation issue is that the ingress nodes need to keep track of the delay differential and path switches for all source-destination pairs in the OBS network, which presents scaling problems. Because of these issues, we evaluate HS-RR-TCP, a higher layer approach, to solve the reordering problem.

IV. RR-TCP

In this section, we provide a brief overview of RR-TCP. For further details refer to [12]. RR-TCP utilizes TCP with DSACK; DSACK is an extension to TCP SACK where the receiver reports the receipt of duplicate segments to sender. A duplicate segment implies that both the original and the retransmitted packet have arrived at the receiver. Using this feedback, the sender can detect that false fast retransmission has occurred.

A. RR-TCP Algorithm

The RR-TCP algorithm helps desensitize the TCP sender to packet reordering by dynamically adjusting the sender’s *dupthresh* parameter. It not only increases the parameter when it detects reordering, but it also must decrease it when necessary in order to prevent timeouts. As *dupthresh* gets larger, small amounts of real loss that would normally result in a fast retransmit will instead result in a timeout because TCP will ignore the duplicate acknowledgments thinking it is reordering occurring. RR-TCP has to balance the tradeoff between false fast retransmits caused by reordering and real timeouts from loss.

RR-TCP works by using the state information maintained by TCP SACK. The two main components of RR-TCP are measuring the individual reordering lengths and maintaining the reordering length samples in a histogram. When an ACK is not received for a segment, there is a hole in SACK’s scoreboard. This missing ACK will cause a fast retransmit and TCP will retransmit that packet. If that packet was reordered

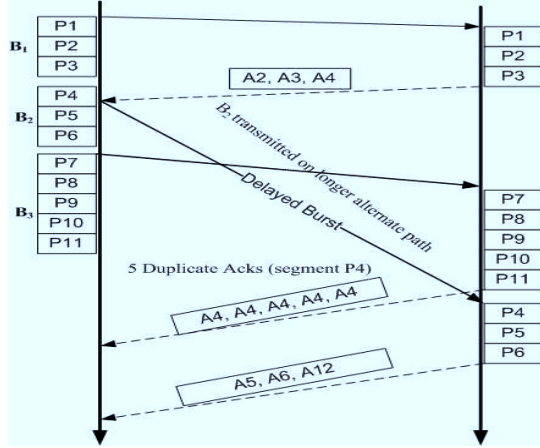


Fig. 1. False fast retransmit example.

instead of lost, TCP will receive two ACKs for it (assuming ACKs are not lost). This is detected with the receipt of a DSACK and the reordering length is measured by taking the distance from the hole in the scoreboard to the cumulative ACK received. The reordering length is then stored in a histogram. Note, if a DSACK is not received, the reordering length is not recorded.

The histogram consists of bins that represent reordering lengths. Every time a reordering length is measured after the receipt of DSACKs, that bin's value is incremented. The samples are removed from the histogram after a specified period of time, which we refer to as the histogram history length.

In order to adjust the *dupthresh* value, the percentage of reorderings to be avoided is selected. This percentage is called the False Fast Retransmit Avoidance ratio, or FA ratio. For example, if 90% of reorderings are to be avoided, it will return the *dupthresh* value that accounts for 90% of the reordering length samples in the histogram.

As previously mentioned, *dupthresh* must be dynamically increased to reduce false fast retransmits and decreased in the case of actual loss to avoid real timeouts. The authors in [12] develop cost functions that are used to increase or decrease the FA ratio whenever a false fast retransmit is detected or a timeout occurs. The functions measure the cost of true timeouts and of false fast retransmissions and are reproduced here:

$$C(\text{true timeout}) = W \left(\frac{T}{R} + \log_2(W - k - 2) + 1 \right).$$

$$C(\text{false fast retransmit}) \leq \frac{k(W - k + 1)}{2}.$$

where W is the steady state window size, R is the smoothed RTT, T is the retransmission window, and k is the limited transmit parameter. Based on these cost functions, the FA ratio is increased by S , a RR-TCP parameter, for every false fast retransmit. Upon every timeout, the FA ratio is decreased by

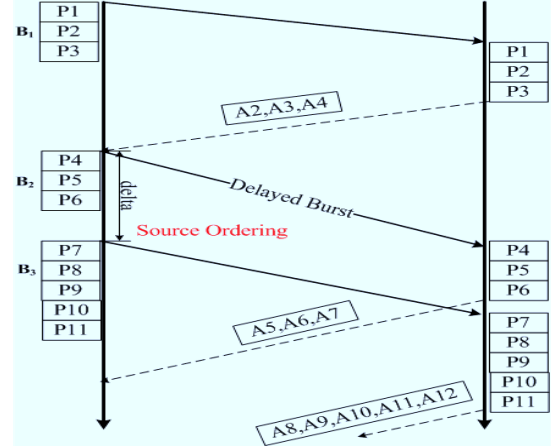


Fig. 2. Load-balanced OBS with source ordering.

$S * C(\text{timeout}) / C(\text{false fast retransmit})$. This algorithm is called DSACK with Timeout Avoidance, or *DSACK-TA*.

In addition to dynamically adjusting the FA ratio, another important feature of RR-TCP is that when a false fast retransmission is detected, it restores the congestion window to the previous value.

B. Illustration

Consider a TCP flow with its *dupthresh* value initially set to 3, which means when a sender receives 3 duplicate ACKs it enters the fast retransmission phase. We will assume all TCP segments in each burst belong to a single TCP flow. In the scenario in Fig. 1, we can see that B_1 is successfully sent and received. B_2 is now sent on the longer alternate path due to load-balancing, while B_3 is sent on the shorter path. These bursts will arrive out of order, leading to out-of-order delivery of packets to the TCP receiver. TCP receiver will send 5 duplicate ACKs when it receives the segments in B_3 , which causes the sender to enter false fast retransmission resending segments 4, 5 and 6 and creates three holes in the sender's scoreboard, one for each of the segments 4, 5, and 6. The sender then receives the cumulative acknowledgement when the ACKs for B_2 are received. It will then receive 3 DSACKs after the retransmitted packets reach the destination. Upon receipt of the DSACKs, there are three reordering length samples stored in the histogram: 5, 6, and 7, one for each segment. Now the values for histogram bins 5, 6, and 7 will be 1 while all other bins are 0.

V. SIMULATION RESULTS

In this section we will discuss simulation results obtained from ns2 with the OWns module [14] for simulating OBS networks. We evaluate HS-RR-TCP over an OBS network under a number of different scenarios with load-balanced routing and then compare HS-RR-TCP to source ordering. The load-balanced routing has two fixed paths and the path chosen is the path with the least congestion. First, we vary the delay differential between the primary and alternate path. Next we evaluate HS-RR-TCP with different burst sizes. After that we examine the impact of loss on HS-RR-TCP. The affects of

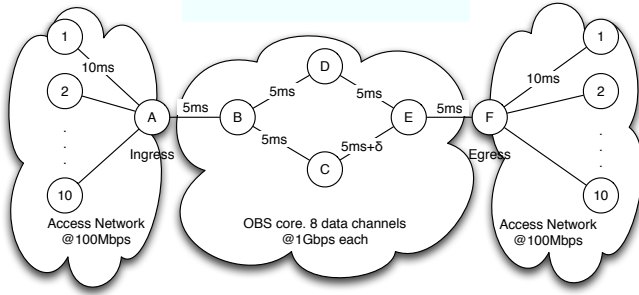
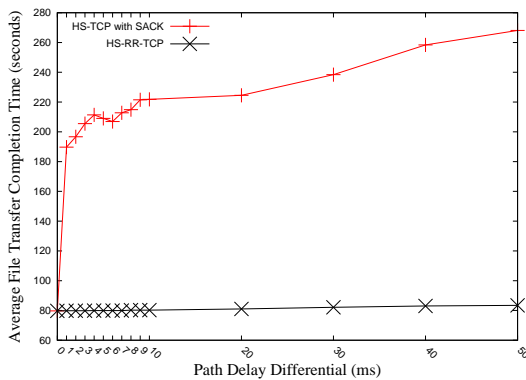
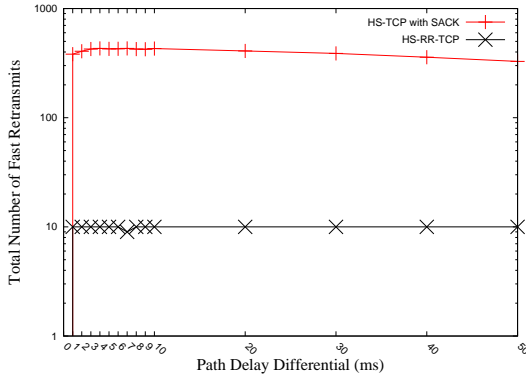


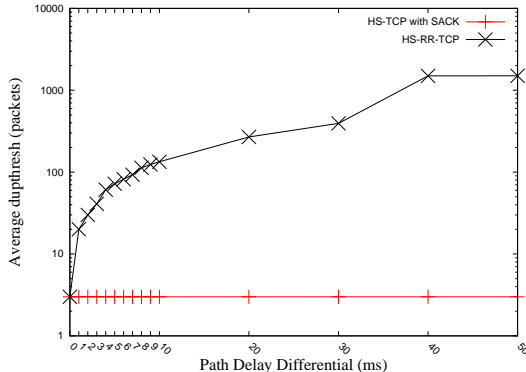
Fig. 3. Simulation topology.



(a) Average file transfer completion time.



(b) Total number of fast retransmits across all flows.



(c) $dupthresh$ for varying delay differentials.

Fig. 4. Comparison of performance of FTP file transfers, each of the 10 TCP flows sending a 1GB file, with varying delay differential.

the load balancing parameter ρ_{max} are investigated as well. We also find the optimal histogram history length used to determine how long samples are kept in the histogram, and the optimal FA ratio. Finally, we make comparisons to source ordering.

The topology used in the simulations is shown in Fig. 3. We have an access network consisting of 10 nodes connected to the OBS ingress node, node A. Each access node has a TCP flow sending to the corresponding node on the right side. The electronic nodes are numbered while the OBS nodes use letters. The primary path in the network is A-B-D-E-F while the alternate path is A-B-C-E-F.

All TCP flows use the High Speed TCP window increase and decrease functionality [13] with SACK which we refer to as HS-RR-TCP and HS-TCP-SACK for TCP with and without the RR-TCP algorithms, respectively. Each flow sends a 1GB file using FTP. The network uses load balancing between the two paths in the core. The τ parameter is set to 500ms and ρ_{max} is set to 5%. This means that every 500ms the path will change if the congestion on the current path exceeds 5%.

For HS-RR-TCP, we use the DSACK-TA algorithm, which both dynamically increases $dupthresh$ and decreases it for timeout avoidance. The default parameters are as follows: HS-RR-TCP uses an 80 second histogram history for keeping samples in the histogram, the max burst size is 100KB, the BAT is 10ms, the delay differential, δ , is 50ms, the FA ratio is 90%, and there is no loss. Each of these parameters (except BAT) will be varied in the following subsections while analyzing the performance of HS-RR-TCP.

A. Performance with varying delay differential (δ)

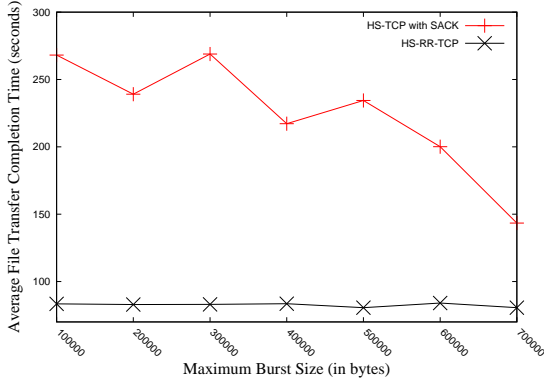
In the first set of simulations we vary the delay differential between the primary and alternate path. There is no loss for these simulations. The results are shown in Fig. 4.

From Fig. 4(a) we observe that even a difference of 1ms in the alternate paths results in reordering. HS-RR-TCP is able to adjust $dupthresh$ to account for the reordering but HS-TCP-SACK experiences false fast retransmissions whenever reordering occurs, resulting in much higher completion times. Fig. 4(b) shows that each of the 10 flows of HS-RR-TCP experience a single false fast retransmit and then adjust their $dupthresh$ value so as to avoid the rest, while HS-TCP-SACK experience false fast retransmits repeatedly. In the case of no loss, we observe up to an 300% improvement in average completion time.

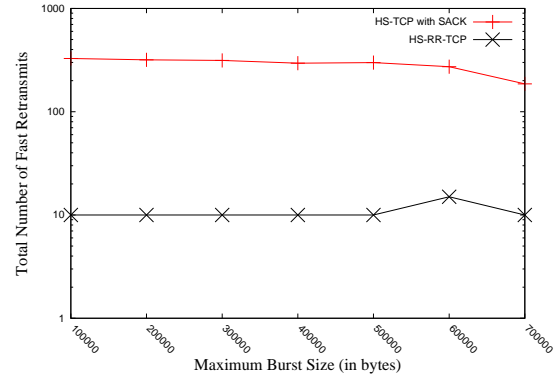
In Fig. 4(c) shows the average value of TCP's $dupthresh$. We observe that as the delay differential increases, larger and larger reorderings, leading to larger $dupthresh$ values, occur. Note, this is with no actual loss, so $dupthresh$ is never decreased. We will look at the scenario with loss in the following subsections.

B. Performance with varying burst size

In this section, we briefly analyze the affects of burst size on reordering. In Fig. 5(a) we plot the completion time while varying the maximum burst size. The burst size has little affect

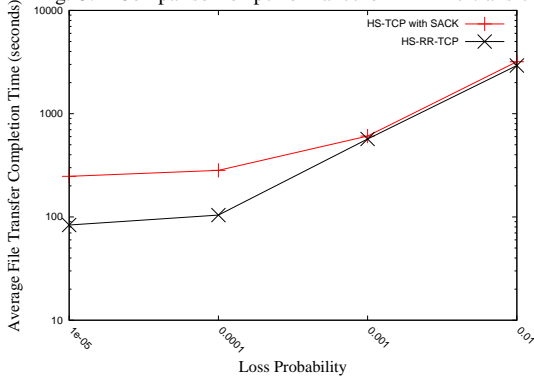


(a) Average file transfer completion time.

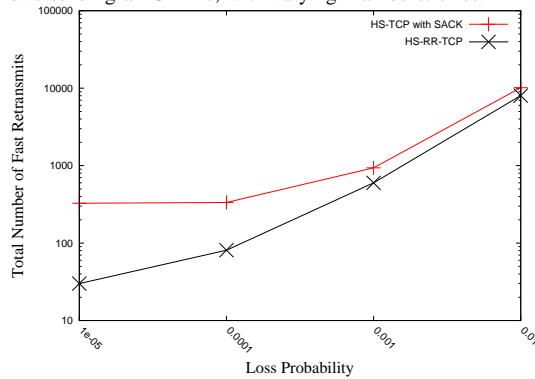


(b) Total number of fast retransmits across all flows.

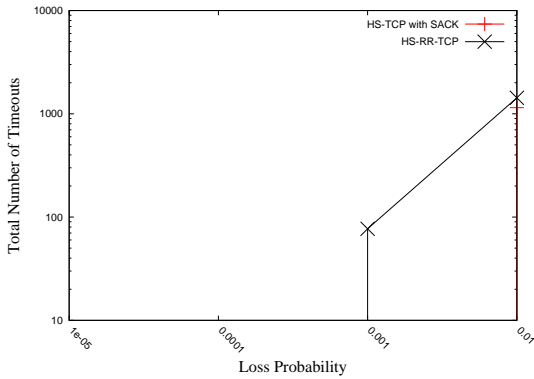
Fig. 5. Comparison of performance of FTP file transfers, each of the 10 flows sending a 1GB file, with varying max burst sizes.



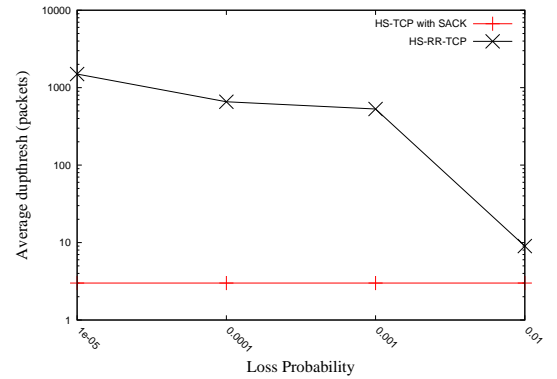
(a) Average file transfer completion time.



(b) Total number of fast retransmits across all flows.



(c) Total number of timeouts across all flows.



(d) Average *dupthresh* value across all flows.

Fig. 6. Comparison of performance of FTP file transfers, each of the 10 flows sending a 1GB file, with random contentions.

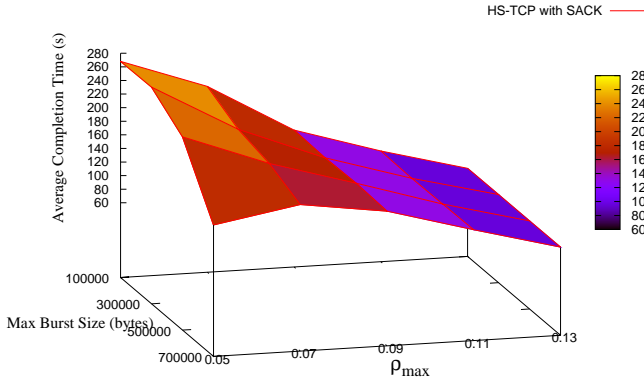
on HS-RR-TCP but does have an affect on normal HS-TCP-SACK. From Fig. 5(b), there is a decrease in the number of false fast retransmits experienced by HS-TCP-SACK as the burst size increases. This is simply because as the bursts get bigger, more data is able to be sent on the same path instead of getting split onto different paths.

C. Performance with varying loss levels

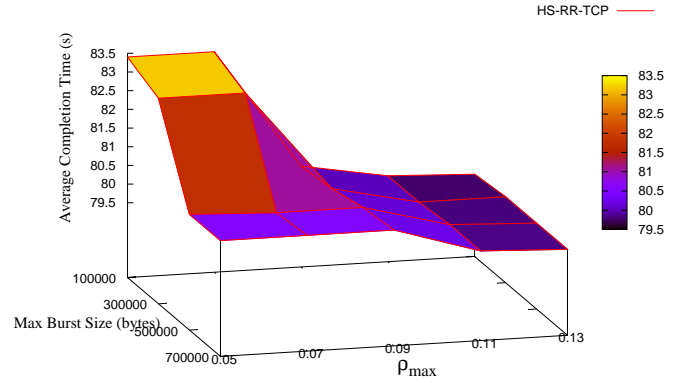
In this section we analyze the affects of random loss in the OBS core on HS-RR-TCP's performance. Fig. 6(a) shows the average completion time for HS-TCP-SACK and HS-RR-TCP. We can see that for low loss probability there is a significant increase in performance, up to 300%, but as loss probability increases, there is little gain. This is due to two fundamental reasons. First, because of real loss, the *dupthresh*

value is being decreased, and second, because of real loss path-switching does not happen as frequently at higher loss probabilities because we only have TCP traffic in the network which reduces its send rate with loss.

The interesting point on this graph is the performance at 0.001 loss probability. This is the last point where there is still reordering in the network, at higher loss probabilities there is not enough TCP traffic to cause reordering (we note here that even at high loss without reordering HS-RR-TCP does not hurt performance). There is a 60s difference in completion time, or about a 6% improvement. Fig. 6(c) shows that only HS-RR-TCP has timeouts at this level of loss. This is because the increase in *dupthresh* is causing timeouts. At the same time, in Fig. 6(b) we can see that it is still reducing the total number of

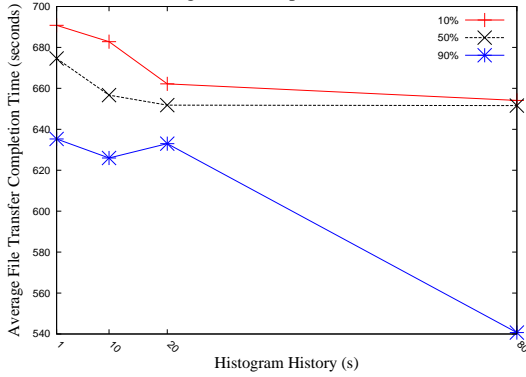


(a) HS-TCP-SACK with varying ρ_{max} values.

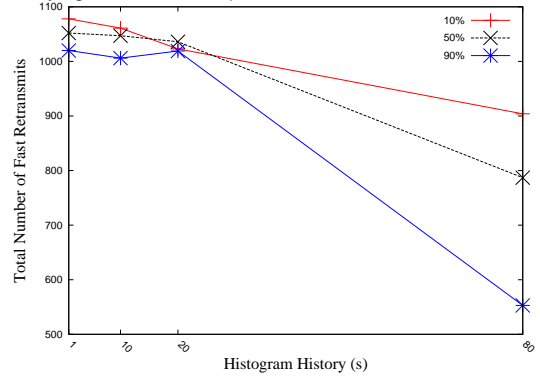


(b) HS-RR-TCP with varying ρ_{max} values.

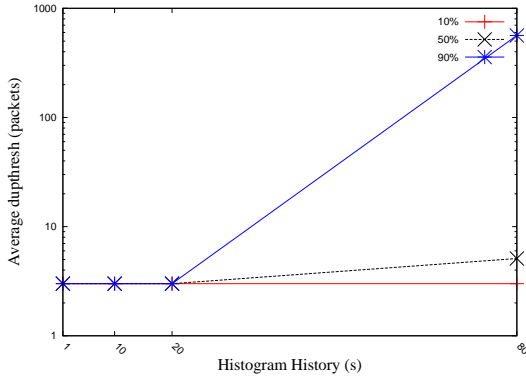
Fig. 7. Comparison of HS-RR-TCP and HS-TCP-SACK for varying burst sizes and ρ_{max} values.



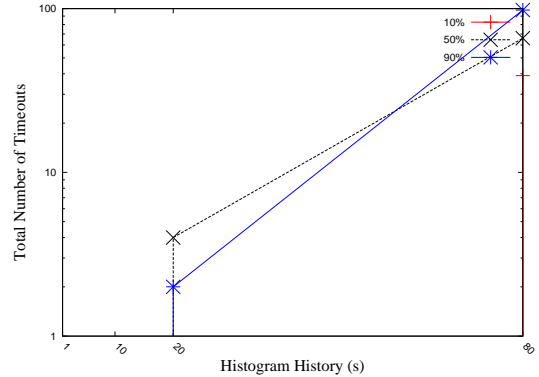
(a) Average file transfer completion time.



(b) Total number of fast retransmits across all flows.



(c) Average *dupthresh* value.



(d) Total number of timeouts across all flows.

Fig. 8. Comparing HS-RR-TCP performance with varying FA ratio and history at 0.001 loss.

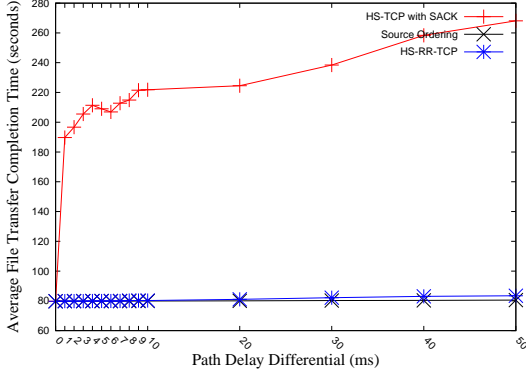
fast retransmissions. There are 330 fewer fast retransmissions and 77 more timeouts using HS-RR-TCP compared to HS-TCP-SACK at 0.001 loss. According to HS-RR-TCP's cost functions, it decides to keep TCP's *dupthresh* value high even though it is causing some timeouts. Looking at Fig. 6(d) we can see that once the loss gets higher, the *dupthresh* is reduced. HS-RR-TCP's cost function is working properly here because even though HS-RR-TCP is causing timeouts, it still has slightly better performance than HS-TCP-SACK.

We also note that when we introduced loss in the access network, we obtained similar results (not shown due to space

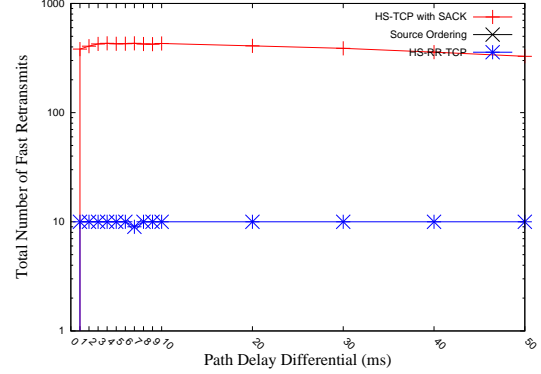
limitations).

D. Performance with varying ρ_{max} values

In this section we analyze the performance impact of the frequency of reordering. We do this by modifying the ρ_{max} parameter to the load balancing algorithm, which determines when a link is congested and therefore switches paths. Fig. 7(a) shows the performance of HS-TCP-SACK varying both the burst size and the ρ_{max} parameter. From the figure we can see that as ρ_{max} increases, performance is also increased. This is because with higher ρ_{max} values, there are fewer path switches and therefore less reorderings. The maximum traffic into the



(a) Average file transfer completion time.



(b) Total number of fast retransmits across all flows.

Fig. 9. Comparison of HS-RR-TCP and Source Ordering with varying δ .

OBS network is 1 Gbps (10 X 100Mbps) and with a maximum capacity of 8 Gbps, the ρ_{max} value of 0.13 represents the best case performance with no reordering (no load-balancing).

Fig. 7(b) shows HS-RR-TCP's performance with varying ρ_{max} values. There is only a small increase in completion time given the different ρ_{max} values. The largest difference occurs at a maximum burst size of 100KB with about a 4s difference between the ρ_{max} values at opposite ends, while at the same time for HS-TCP-SACK there is an almost 190s difference. Clearly, HS-RR-TCP obtains close to optimal performance.

HS-RR-TCP and HS-TCP-SACK have the same completion times at $\rho_{max} = 0.13$ (about 79.5s), so HS-RR-TCP does not hurt performance when there is no load balancing. We can also see that with HS-RR-TCP, burst size has little affect on performance.

E. Performance with varying histogram history lengths and FA ratios

In the previous subsections we have used the 90th percentile for the FA ratio and we also let samples stay in the histogram for 80 seconds. In this section we show the results of trying different values at varying loss levels since loss has a significant impact on HS-RR-TCP's performance.

We compare the parameter at the loss probability 0.001. At lower and no loss all of the different parameter settings performed very similarly and at higher loss there is little or no reordering.

Fig. 8 compares the HS-RR-TCP parameters for history and FA ratio at 0.001 loss. From Fig. 8(a) it is clear that the 90th percentile and a longer history outperform the other combinations. This is the combination where the *dupthresh* value raises from the default, as shown in Fig. 8(c). Because of the large amount of time it stores history and the high percentile, it is able to increase *dupthresh* even though there are fewer reorderings due to higher loss.

This higher *dupthresh*, while reducing fast retransmissions as seen in Fig. 8(b) also increases timeouts as seen in Fig. 8(d). We saw this result previously in Section V-C. Even though there are more timeouts, the difference between number of fast retransmissions is greater than the number of timeouts, so performance is still better. The cost function is working properly.

From these results, we conclude that a larger history, 80s, and a higher FA ratio, 90%, are optimal values.

F. Comparison of HS-RR-TCP and Source Ordering

In this section we include source ordering in our results for varying delta and loss to compare the performance of HS-RR-TCP and source ordering. Fig. 9 (a) shows that there is very little improvement using OBS-layer source ordering instead of HS-RR-TCP. The difference increases slightly as the delay differential increases, but at $\delta = 50ms$ there is only a difference of a few seconds, and path delay differentials greater than this are not realistic. From Fig. 9 (b) we observe no false fast retransmits experienced using source ordering and only one for each flow for HS-RR-TCP.

We also ran source ordering simulations for varying loss levels, the results are shown in Fig. 10. Fig. 10(a) shows that source ordering performs similar to HS-RR-TCP at low loss but then performs the same as HS-TCP-SACK at higher loss values. At higher loss there is very little reordering so the two mechanisms described to prevent it have little affect. At 0.001 and 0.01 loss source ordering actually performs worse than HS-TCP-SACK. This may be because that with the lack of reordering the simulations without source ordering are sending on the primary path with shorter delay while source ordering is always sending data over a path with longer delay (either the longer path or the shorter path plus buffering). Fig. 10(b) shows very similar number of fast retransmission between source ordering and HS-TCP-SACK at higher loss indicating the performance difference is because of the extra δ used for buffering with source ordering.

From these results, we can see that source ordering provides limited benefit compared to HS-RR-TCP. Source ordering requires adding electronic buffers at the ingress nodes while HS-RR-TCP is a transport layer mechanism that modifies the SACK extension and does not increase asymptotic computation or storage complexity of SACK. Resolving reordering using HS-RR-TCP seems to be a better choice.

VI. CONCLUSION

In this paper we have examined the performance of HS-RR-TCP over a load-balanced optical burst-switched network. We have found that HS-RR-TCP is able to reduce the number

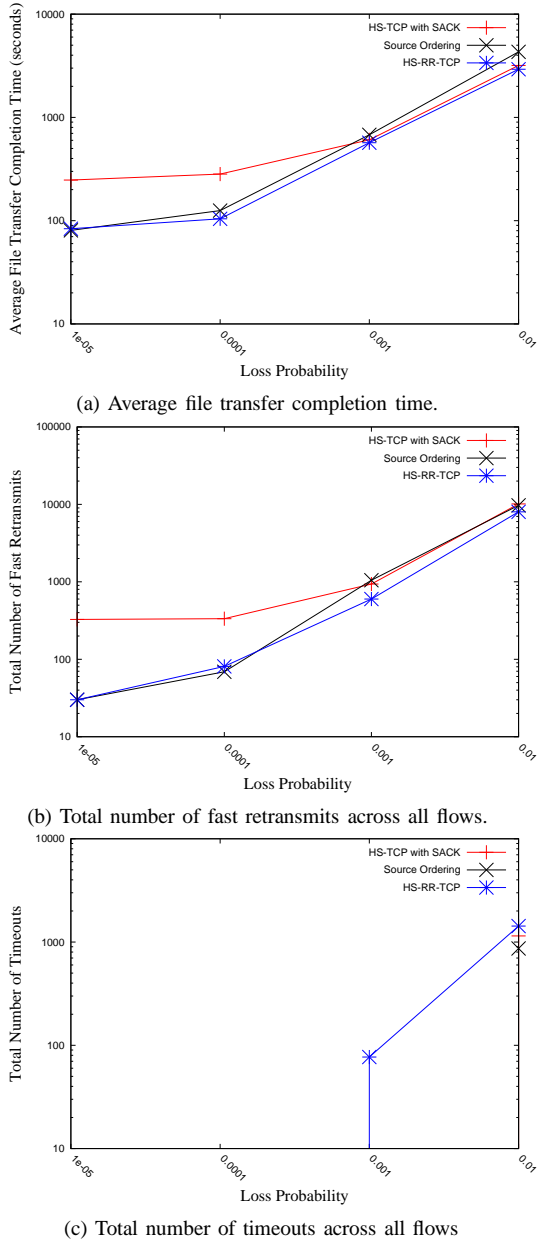


Fig. 10. Comparison of HS-RR-TCP and Source Ordering with varying loss.

of false fast retransmits experienced by TCP caused by re-ordering of bursts during load-balancing. With low actual loss (which causes more frequent reorderings) we see performance is almost tripled for average file transfer completion time. As loss increases, HS-RR-TCP still provides a performance enhancement.

We compared HS-RR-TCP to an OBS layer solution called source ordering. While source ordering can provide slight performance increases over HS-RR-TCP, it requires electronic buffers at the ingress nodes while HS-RR-TCP is only a transport layer modification.

Areas of future work include evaluating the use of HS-RR-TCP for other scenarios, such as reducing the affects of reordering caused by contention resolution mechanisms like deflection routing and burst retransmission as identified

in [15]. We are currently developing an analytical model that computes the file transfer completion time for TCP over load-balanced OBS.

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, Jan. 1999.
- [3] I. Stoica and et. al., "Chord: A scalable peer-to-peer lookup protocol for Internet applications," in *Proceedings of ACM SIGCOMM*, 2001.
- [4] 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.
- [5] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the Grid: enabling scalable virtual organizations," *International Journal of High Performance Computing Applications*, vol. 15, pp. 200–222, 2004.
- [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*, Mar. 2003.
- [7] 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, Nov.-Dec. 2003.
- [8] 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*, Dec. 2003, vol. 5, pp. 2694–2698.
- [9] J. Li, M. Gurusamy, and K. C. Chua, "Load balancing using adaptive alternate routing in IP-over-WDM optical burst switching networks," in *SPIE Optical Networking and Communications (OptiComm)*, 2003, vol. 5285, pp. 336–345.
- [10] Li Yang and G.N. Rouskas, "Adaptive path selection in OBS networks," in *IEEE/OSA Journal of Lightwave Technology*, vol. 24, no. 8, pp. 3002–3011, Aug. 2006.
- [11] B. Komattireddy and V.M. Vokkarane, "Source-ordering for improved TCP performance over load-balanced optical burst-switched (OBS) networks," in *Proceedings, IEEE BroadNets 2007*, Sep. 2007.
- [12] M. Zhang, B. Karp, S. Floyd, and L. Peterson, "RR-TCP: A reordering-robust TCP with DSACK," in *Proceedings, IEEE International Conference on Networking Protocols (ICNP 2003)*, Nov. 2003, pp. 95–106.
- [13] S. Floyd, "Highspeed TCP for large congestion windows," RFC 3649, Dec. 2003.
- [14] "OBS-NS simulator: <http://wine.icu.ac.kr/obsns/index.php>," .
- [15] J. Perello, S. Gunreben, and S. Spadaro, "A quantitative evaluation of reordering in OBS networks and its impact on TCP performance," *International Conference on Optical Network Design and Modeling (ONDM) 2008*, pp. 1–6, Mar. 2008.