Optical Switching and Networking 8 (2011) 116–128

Contents lists available at ScienceDirect

Optical Switching and Networking

journal homepage: www.elsevier.com/locate/osn

Performance modeling of HS-RR-TCP over load-balanced optical burst-switched (OBS) networks[☆]

Neal Charbonneau, Vinod M. Vokkarane [*]

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

ARTICLE INFO

Article history:
Received 2 July 2009
Received in revised form 15 June 2010
Accepted 14 October 2010
Available online 26 October 2010

Keywords:
Load-balanced routing
TCP-SACK
HS-TCP
OBS

ABSTRACT

TCP-over-OBS is a promising transport paradigm to support next-generation Internet. It is well known that 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 the performance of Reordering-Robust TCP (RR-TCP) over OBS networks. We develop an analytical end-to-end transfer delay model for TCP SACK and RR-TCP over load-balanced OBS networks. We observe that standard TCP experiences significant throughput degradation due to persistent packet reordering. Through simulations and modeling we show that RR-TCP achieves significant improvement compared to TCP SACK. In our simulations we add High-Speed TCP functionality to RR-TCP (HS-RR-TCP). The simulations show up to a 300% performance improvement for HS-RR-TCP compared to HS-TCP-SACK under ideal conditions and a 20% improvement on NSFnet with background traffic.

1. 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-switched (OBS) networks, data to be transmitted is assembled into bursts that 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 bursts' route. In OBS networks, apart from the data channels, each link has one or more control channels to transmit BHPs. BHPs carry 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. The data burst then 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.

[☆] This work was supported in part by the National Science Foundation (NSF) under grant CNS-0626798.
[*] Corresponding author. Tel.: +1 5089106692; fax: +1 508 999 9144.
E-mail addresses: ncharbonneau@ieee.org (N. Charbonneau), vvokkarane@ieee.org, vvokkarane@umassd.edu (V.M. Vokkarane).

Due to the bufferless nature of the 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 sender's 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 first type is known as a fast source while the second type is known as a medium source [6].

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 [7], and burst segmentation [8]. 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 [9–11]. 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 load-balanced routing with static route-calculation and dynamic route-selection as proposed in [9]. 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 $\rho_{\max}$ 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 links is expressed as the duration of all arriving bursts over the interval τ, and is given by, $L_{i,j} = \frac{\tau_s+\tau_d}{\tau}$. Load-balancing leads to persistent reordering of bursts that degrade TCP's performance (for details refer to Section 2).

It was shown in [12] 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. Source ordering is an OBS-layer technique for solving the burst reordering problem. When a path switch is made to a path with shorter delay, the bursts are buffered long enough to prevent them from arriving before bursts sent previously on the longer path. It was shown that TCP's performance with source ordering and load-balancing was better than TCP over an 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 along with electronic buffers at the OBS ingress nodes. In this paper we aim to resolve the reordering independently at the higher TCP-layer. Reordering-Robust TCP (RR-TCP) has been proposed [13] 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 [14]) over a load-balanced OBS network under different network conditions and also develop analytical models of RR-TCP and TCP SACK over a load-balanced OBS network.

The remainder of the paper is organized as follows. Section 2 discusses the issues of TCP over a load-balanced OBS network. Section 3 describes the *RR-TCP* mechanism, then Section 4 presents our analytical end-to-end transfer delay model of TCP SACK and RR-TCP. Section 5 provides our numerical results. Lastly, Section 7 concludes the paper.

2. TCP over load-balanced OBS

In a load-balanced network, each ingress uses two 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 $\rho_{\max}$, 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 a 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 (FR) 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 the 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* (FFR).

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 [14] with SACK and RR-TCP to better utilize the high-speed OBS network. This means we apply HS-TCP's modified congestion control and integrate it with RR-TCP.

3. RR-TCP

In this section, we provide a brief overview of RR-TCP. For further details refer to [13]. RR-TCP utilizes TCP with

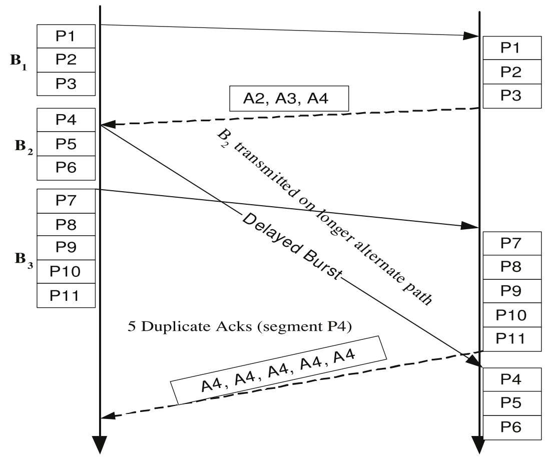


Fig. 1. False fast retransmit example.

DSACK; DSACK is an extension to TCP SACK where the receiver reports the receipt of duplicate segments to the 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 FFR has occurred.

3.1. RR-TCP algorithm

The RR-TCP algorithm helps desensitize the TCP sender to packet reordering by dynamically adjusting the sender's *dupthresh* parameter. The TCP sender not only has to increase the *dupthresh* when the sender detects reordering, but the sender must also decrease *dupthresh* 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. This is due to the fact that TCP will ignore the duplicate acknowledgments assuming the packets have been reordered. RR-TCP has to balance the tradeoff between FFRs caused by reordering and real timeouts from actual 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 instead of being 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.

The reordering samples in the histogram are used to adjust the *dupthresh* value. The value of *dupthresh* is set based on the percentage of reorderings causing FFRs to avoid. For example, if 90% of reorderings are to be avoided, it will return the *dupthresh* value that represents that percentile value in the cumulative reordering length distribution. This percentage is called the **F**alse Fast Retransmit **A**voidance ratio, or FA ratio.

As previously mentioned, *dupthresh* must be dynamically increased to reduce FFRs and decreased in the case of actual loss to avoid real timeouts. The authors in [13] have developed cost functions that are used to increase or decrease the FA ratio whenever a FFR is detected or a timeout occurs. The functions measure the cost of true timeouts (TTO) and the cost of FFRs. They are reproduced here:

$$C(\mathrm{TTO}) = W\left(\frac{T}{R} + \log_2(W-k-2) + 1\right),$$

$$C(\mathrm{FFR}) \le \frac{k(W-k+1)}{2},$$

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 $S * C(\mathrm{TTO})/C(\mathrm{FFR})$. This algorithm is called DSACK with Timeout Avoidance (*DSACK-TA*).

In addition to dynamically adjusting the FA ratio, another important feature of RR-TCP is that when an FFR is detected, it restores the congestion window to the value before the FR.

3.2. 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 sender. 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. The 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 B_2 is 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.

4. Analytical modeling

In this section we develop end-to-end transfer delay models for both TCP SACK and RR-TCP over a load-balanced OBS network. We assume that there is bifurcation of traffic in the network at the burst level, so each bust sent is

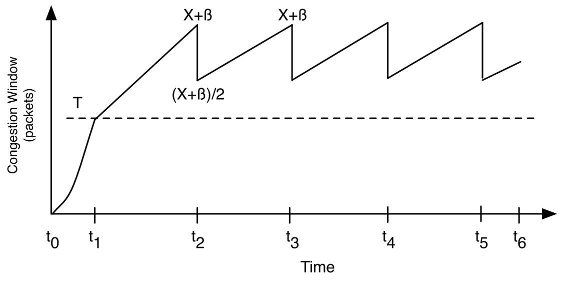


Fig. 2. Window growth for TCP SACK over LB-OBS.

alternated on two link-disjoint paths. For the model, we do not consider an access network. We assume a high enough bandwidth on the access network to allow the entire TCP sender's window to be placed in a burst before the burst assembly timer (BAT) times out. This also means that our model is for TCP fast flows as described in [6]. The model represents a simplified scenario to show the impact of reordering on TCP SACK and the improvement that can be gained by using RR-TCP in an ideal scenario. The modeling allows us to understand what is happening to a traditional TCP flow as the paths change constantly. We use simulation for more realistic and complicated scenarios in Section 5.

To model the delay, we consider four phases of a file transfer. The congestion window growth of a TCP SACK flow can be seen in Fig. 2. The first phase is the initial slow-start phase between t_0 and t_1. The second phase is the time leading up until the first false loss detection, between t_1 and t_2. After t_2 there are a number of periods where the sender's congestion window is halved because of false loss detection caused by triple duplicate ACKs from re-ordered bursts. We will refer to these periods as *false triple duplicate periods* (FTDPs). An example of a FTDP is between t_2 and t_3. The third phase consists of all the FTDPs (t_2–t_5) and lastly the final phase is comprised of the remainder of the file transfer after the last FTDP (t_5–t_6).

Assuming a long lived TCP flow, the window will grow to size T, which is the slow-start threshold, by time t_1. Let X be the maximum burst size ($X \geq T$). When the congestion window reaches size X, the whole window still fits into a single burst. As the window increases, two bursts will be sent, which will eventually lead to reordering. Let the window size that causes reordering be $X + \beta$, which is shown at time t_2. The value for β, which we will derive, depends on the delay differential between the two paths. Time t_2 starts the first FTDP. Note, that after the fast retransmit, the slow start threshold, T, is adjusted so the flow will not enter SS and grow its window exponentially anymore. The remainder of the transfer will be a number of FTDPs. We refer to the congestion window size that causes reordering to be M, where $M = X + \beta$.

The model will consist of four phases described previously. The first being slow-start (t_0–t_1), the next being the phase leading up to the first FTDP (t_1–t_2), then the FTDPs (t_2–t_5), and lastly the remainder of the transfer (t_5–t_6). We base our end-to-end transfer delay model on the TCP delay model presented in [15].

We will use the following notation and assumptions:

t_a: burst assembly time.
t_o: burst offset time.
d_s: propagation delay on the shorter path in the OBS network.
t_t: burst transmission delay.
δ: delay differential between the shorter and longer path.
R: bandwidth of a wavelength (all links/wavelengths assumed to have same bandwidth).
T: slow-start threshold (in packets).
X: maximum burst size (in packets).
M: burst size that causes a reordering ($T \leq X \leq M$).
S: TCP segment size (assuming segment size is fixed).

We assume that delayed ACKs are not implemented and ACKs always take the shorter path, so the RTT (based only on propagation delay of the path) for the shorter path is $2d_s$. The one-way delay for the longer path is $d_s + \delta$, so the RTT is $2d_s + \delta$. We assume that the slow-start threshold of the TCP connection is smaller than the maximum burst size so this phase will be completed before the window is split across multiple bursts. The actual RTT of the paths depends not only on the propagation delay but also OBS specific properties like the assembly time and offset times. We assume JET signaling in the core, so the total RTT for the shorter path is $2t_a + 2t_o + 2d_s + t_t$ while the total RTT for the longer path is $2t_a + 2t_o + (2d_s + \delta) + t_t$ as described in [16]. t_t depends on the burst size and the fiber bandwidth, R $\left(t_t = \frac{\text{burst size}}{R}\right)$, while t_o is calculated based on the path length, BHP processing time, and switching time.

4.1. TCP SACK modeling

As we previously discussed, we will calculate the entire end-to-end transfer delay for TCP SACK by calculating the delay on the individual phases of the file transfer. When we say delay, we mean the time it takes TCP SACK to complete the current phase. The delay for the slow-start period is how long TCP SACK will spend in slow-start. We will refer to the delay for the slow-start period as Delay_{SS}. The delay for the slow-start phase consists of 2RTT*s* for the TCP connection setup in addition to the time it takes the TCP sender's window to reach the slow-start threshold. Each sending round consists of an *idle period*, which is the time it takes to transmit the burst and receive the ACKs from the TCP receiver (allowing the sender to begin the next round). We can define the idle period for the k-th slow-start round as,

$$\text{idle period}_k = 2t_a + 2t_o + 2^{k-1}S/R + \text{Delay}_{\text{OBS}} \tag{1}$$

where $\text{Delay}_{\text{OBS}}$ is the propagation delay of the path taken. The assembly time and offset time remain the same, but as the burst gets larger (the TCP sender's window increases), the transmission time ($t_t = 2^{k-1}S/R$) increases. Since we assume burst-level bifurcation of traffic, half of the bursts will take the shorter path while the other half will take the longer path, so $\text{Delay}_{\text{OBS}}$ can be obtained by the average,

$$\text{Delay}_{\text{OBS}} = (4d_s + \delta)/2. \tag{2}$$

There will be P idle periods where $P = \lceil \log_2(T+1) \rceil$, since this is slow-start (see Appendix), therefore,

$$\text{Delay}_{\text{SS}} = 2(2t_a + 2t_o + 2d_s) + \sum_{k=1}^{P} (\text{idle period}_k). \qquad (3)$$

The first term of Delay_{SS} is the RTTs required to setup the TCP connection. Plugging (1) and (2) into (3) we get,

$$\begin{aligned}\text{Delay}_{\text{SS}} &= (2+P)(2t_a + 2t_o) + (4+P)d_s \\ &\quad + P/2(2d_s + \delta) + (2^P - 1)S/R.\end{aligned}$$

Next we define the delay for the phase between slow-start and the FTDPs, which we will define as $\text{Delay}_{\text{PFTDP}}$. At this point, the TCP sender is in congestion avoidance and the window will start at T and grow to M, the maximum burst size before reordering occurs $M = X + \beta$. We will define β shortly.

$$\begin{aligned}\text{Delay}_{\text{PFTDP}} &= \sum_{k=T}^{M} (2t_a + 2t_o + kS/R + \text{Delay}_{\text{OBS}}), \\ &= (M-T)((2d_s+\delta)/2 + d_s + 2t_a + 2t_o) \\ &\quad + (M(M+1)/2 - (T-1)T/2)S/R.\end{aligned}$$

In this equation the value of $t_t = kS/R$ because of congestion avoidance. After this phase, the remainder of the connection will be experiencing a number of FTDPs. We must first derive the value of M, which is the window size that will cause re-ordering. To derive this, assume that the burst containing X packets, where X is the max burst size, is sent on the shorter path initially. The remainder of the window, β, will be sent on the longer path. We need to find the largest value of β before a re-ordering. When a re-ordering will occur depends on the delay differential, δ. It will occur when two bursts taking the shorter path arrive before a burst sent on the longer path does. The number of rounds required for this to happen is given by,

$$\begin{aligned}\beta &= \max\{x : (2x-1)d_s \geq (2(x-1)-1)d_s + (x-1)\delta\}, \\ &= \left\lfloor \left(\frac{2d_s+\delta}{\delta}\right) \right\rfloor.\end{aligned}$$

β looks at when the egress node will receive two bursts from the shorter path before a burst comes in on the longer path. x represents the round number. This gives us a window size of $X + \beta$, or M, that will cause the re-ordering. It may be the case that δ will allow the window to take up two bursts without re-ordering. When the window is split across three bursts, then re-ordering will occur immediately, the value of β should take this into account:

$$\beta = \min\left\{ \left\lfloor \left(\frac{2d_s+\delta}{\delta}\right) \right\rfloor, X \right\}.$$

Note, in real implementations of TCP, the window may be rounded up when receiving the larger burst before the burst containing a single packet, resulting in an additional burst to be sent. We found that the above calculation for β to be a good estimate.

Now we will determine the delay for all of the FTDPs, $\text{Delay}_{\text{FTDP}}$. We now know the maximum TCP sender's window size that will cause re-ordering. We next determine the number of FTDPs the TCP sender will experience while sending the file. First the amount of data sent in one FTDP (in packets) is given by:

$$\sum_{i=0}^{M/2} (M/2) + \sum_{i=0}^{M/2} i + M/2 = 3M^2/8 + 3M/4.$$

Therefore the number of FTDPs, n, that the TCP sender will experience is:

$$n = \left\lfloor \frac{O'}{S * (3M^2/8 + 3M/4)} \right\rfloor$$

where O' is the remaining size of the file after the previous two phases. In each FTDP phase, $(X-\beta)/2$ rounds are sent that have windows fitting into a single burst (the window starts at $(X+\beta)/2$ and goes to X). The remaining β windows are larger than the maximum burst size, so the windows are split across two bursts. While the window is covered by one burst, it is alternating between paths. When the window is split across two bursts for the first time, we assume that the burst on the shorter path has X packets and the other burst has the remaining packets. The delays for the portions of FTDPs that transmit the entire window in a single burst are covered by:

$$\begin{aligned}&\text{Delay}_{\text{SINGLE}} \\ &\quad = n \sum_{i=0}^{(X-\beta)/2} \left(2t_a + 2t_o + \text{Delay}_{\text{OBS}} + S * \frac{M/2+i}{R} \right) \\ &\quad = n\,((X-\beta)(t_a + t_o + d_s/2 + (2d_s+\delta)/4) \\ &\qquad + (S/R)\left(\frac{(X-\beta)*M}{4} + \frac{\left(\frac{X-\beta}{2}\right)\left(\frac{X-\beta}{2}+1\right)}{2} \right) \Bigg). \qquad (4)\end{aligned}$$

This covers the delay of all FTDPs where the window is less than the maximum burst size. After that point, the window will cover two bursts. One burst will always have X packets to send, which is sent β (the number of rounds before reordering occurs) times for each FTDP, so the delay is:

$$\text{Delay}_{\text{MAXSIZE}} = \beta * n * (2t_a + 2t_o + 2d_s + S * X/R). \qquad (5)$$

The delay for the other burst sending the remainder of the window is:

$$\begin{aligned}\text{Delay}_{\text{MINSIZE}} &= n \sum_{i=1}^{\beta} (2t_a + 2t_o + 2d_s + \delta + S * (i/R)), \\ &= n\beta \left(2t_a + 2t_o + 2d_s + \delta + \frac{(\beta^2+\beta)/2}{R} \right). \qquad (6)\end{aligned}$$

We can now define $\text{Delay}_{\text{FTDP}}$ as (4) + min{(5), (6)}, meaning the total delay consists of the delay when the window was less than the maximum burst size ($\text{Delay}_{\text{SINGLE}}$) and when the window was split across two bursts

($\text{Delay}_{\text{MAXSIZE}}$ and $\text{Delay}_{\text{MINSIZE}}$).

We will now derive the delay for the phase after the FTDPs, $\text{Delay}_{\text{FINAL}}$. Let O'' be the remainder of the file that needs to be sent after the final FTDP. To determine how many rounds are required to send the remainder of the file, we must calculate:

$$\min\{j : MS/2 + MS/2 + 1 + MS/2 + 2 + \cdots + MS/2 + j \geq O''\};$$

$$\min\left\{j : \sum_{i=0}^{j}(M/2 + i) \geq O''/S\right\};$$

$$\min\left\{j : jM/2 + \frac{j^2 + j}{2} \geq O''/S\right\};$$

$$j = \frac{-(M+1) + \sqrt{(M+1)^2 - 8O''/S}}{2}. \quad (7)$$

Given j, the delay for the final phase is given by,

$$\begin{aligned}\text{Delay}_{\text{FINAL}} &= \sum_{i=0}^{j}(2T_a + 2T_o + \text{Delay}_{\text{OBS}} + (S * M/2 + i)/R), \\ &= j(2T_a + 2T_o + d_s + (2d_s + \delta)/2) + (S/R)(jM/2 + (j^2 + j)/2).\end{aligned}$$

The total end-to-end data transfer delay is equal to $\text{Delay}_{\text{SS}} + \text{Delay}_{\text{PFTDP}} + \text{Delay}_{\text{FTDP}} + \text{Delay}_{\text{FINAL}}$.

4.2. RR-TCP delay modeling

This model can be modified to model the delay of RR-TCP with 100% FA ratio. With 100% FA ratio, only the first reordering will cause an FFR and the rest will be prevented, as shown in Fig. 3. From the figure, once the FFR has been detected, the window is restored to $X + \beta$ and the reordering length is stored. Modeling RR-TCP is therefore similar to modeling TCP over a regular OBS network. This can be achieved by modifying (7) to determine the number of rounds required after slow-start to the end of the file transfer while in congestion avoidance (we ignore the decrease then restoration of the congestion window since this only takes one round). The updated (7) would be

$$\min\{j : T + (T+1) + (T+2) + \cdots + (T+j) \geq O - (2^P - 1)S\};$$

$$j = -(0.5 + T) + \sqrt{(1 + 2T)^2/4 - (2O/S - 2(2^P - 1))} \quad (8)$$

because congestion avoidance starts at T and will continue until the remainder of the file is sent. O is the original file size and we send $2^P - 1$ packets in slow start. $\text{Delay}_{\text{FINAL}}$ is also modified slightly so that instead of the $M/2$ term we have T, which results in the following equation

$$\text{Delay}_{\text{FINAL}} = j(2T_a + 2T_o + d_s + (2d_s + \delta)/2) + (S/R)(jT + (j^2 + j)/2). \quad (9)$$

The total end-to-end data transfer delay is then equal to Delay_{SS} + (9) (where we use j from (8)).

5. Numerical results

5.1. Analytical model verification

In this section we verify the analytical model developed in the previous section. To verify the model, we use only a

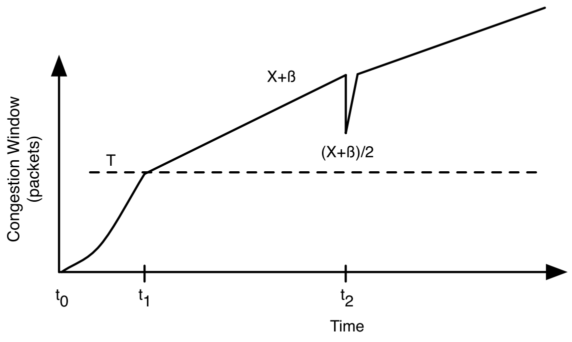


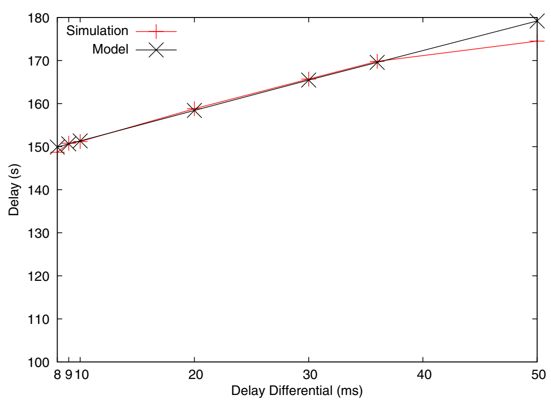
Fig. 3. Window growth for RR-TCP over LB-OBS.

single TCP SACK sender (without High-Speed) and perform bifurcation of the traffic, i.e., each time a burst is sent the path is switched. For the model, the maximum burst size is set to 1000 packets, with slow-start threshold set to 512 packets. The packet size is fixed at 1 kB. A 1 GB file is sent by the single sender. The network used is shown in Fig. 5, except that there is no access network, the TCP sender sits on ingress node A and the receiver on egress node F. The simulation verifying the model is shown in Fig. 4(a). Delay refers to the total time it takes to transmit the file. As the path delay differential increases, the completion time increases because some bursts must take the longer path whose increased propagation delay increases total completion time. The increased delay differential can also cause different amounts of reordering, which will also impact performance.

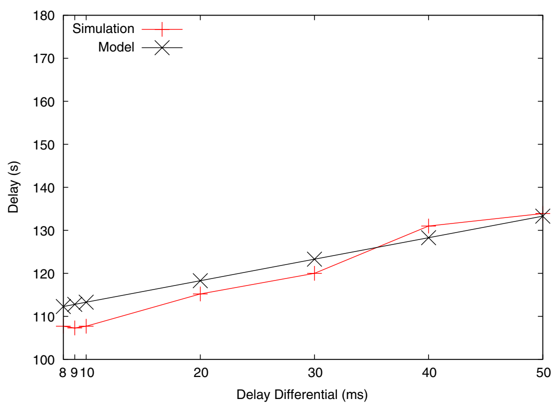
We also show the results for the RR-TCP with 100% FA ratio. The assumptions are the same except that instead of TCP SACK we run RR-TCP. This can be seen in Fig. 4(b). Comparing the two, we can see a significant improvement, especially with larger path delay differentials. This improvement could be larger if the maximum burst size was smaller and therefore lead to more reordering when the window became too big for the burst. We used these settings to make the analysis easier.

5.2. Simulation results

In this section we will discuss simulation results obtained from ns2 with the OWns module [17] 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. Then we analyze the impact of the load-balancing parameters (ρ and τ). We also find the optimal histogram history length used to determine how long samples are kept in the histogram, and the optimal FA ratio. We then compare HS-RR-TCP with the previously proposed source ordering approach. We use a simple topology to make understanding the performance of HS-RR-TCP easier and to save time running simulations. We also run HS-RR-TCP over NSFnet, which we will briefly discuss at the end of the simulation section.



(a) Verification of TCP SACK delay model.



(b) Verification of RR-TCP with 100% FA ratio delay model.

Fig. 4. Verification of analytical model.

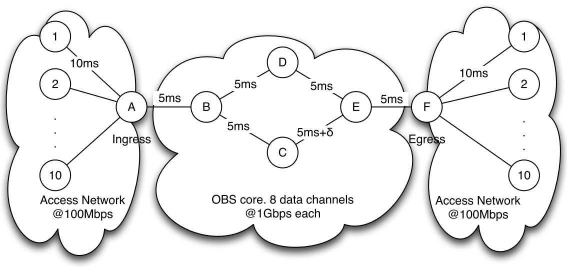


Fig. 5. Simulation topology.

The topology used in the simulations is shown in Fig. 5. 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 [14] 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 1 GB file using FTP, which ensures that the network reaches steady state. The network uses load balancing between the two paths in the core. The τ parameter is set to 500 ms and $\rho_{\max}$ is set to 5%. This means that every 500 ms 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 s histogram history for keeping samples in the histogram, the max burst size is 100 kB, the BAT is 10 ms, the delay differential, δ, is 50 ms, 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.

5.2.1. 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. 6.

From Fig. 6(a) we observe that even a difference of 1 ms 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. 6(b) shows that each of the 10 flows of HS-RR-TCP experiences a single false fast retransmit and then adjusts 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.

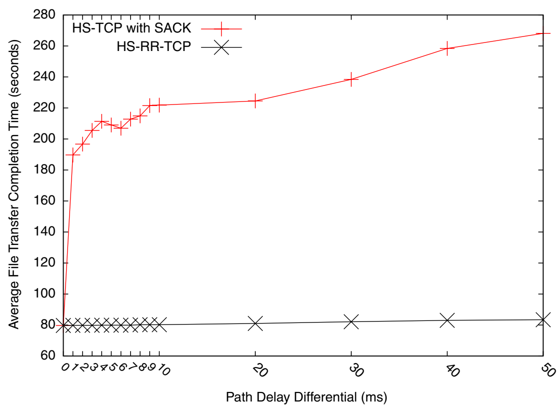
In Fig. 6(c) shows the average value of TCP's *dupthresh*. We observe that as the delay differential increases, larger and larger reorderings, leading to larger *dupthresh* values, occur. Note, this is with no actual loss, so *dupthresh* is never decreased. We will look at the scenario with loss in the following subsections.

5.2.2. Performance with varying burst size

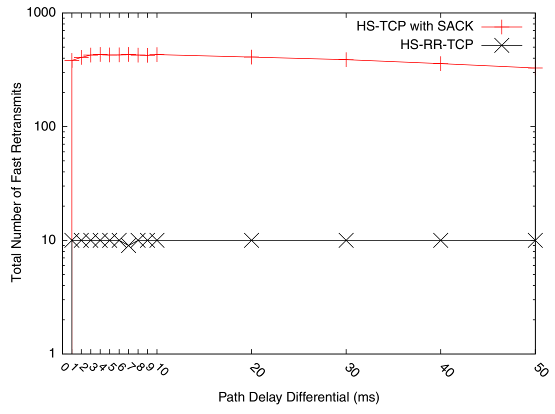
In this section, we briefly analyze the affects of burst size on reordering. In Fig. 7(a) we plot the completion time while varying the maximum burst size. The burst size has little affect on HS-RR-TCP but does have an affect on normal HS-TCP-SACK. From Fig. 7(b), there is a decrease in the number of 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.

5.2.3. Performance with varying loss levels

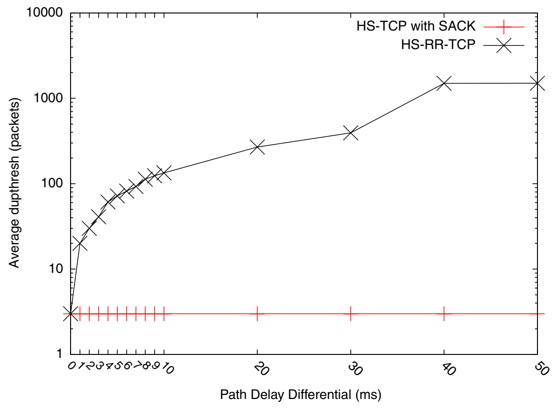
In this section we analyze the affects of random loss in the OBS core on HS-RR-TCP's performance. Fig. 8(a) shows the average completion time for HS-TCP-SACK and HS-RR-TCP. We can see that for low loss probability there is a significant increase in performance, up to 300%, but as loss probability increases, 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.



(a) Average file transfer completion time.



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



(c) *Dupthresh* for varying delay differentials.

Fig. 6. Comparison of performance of FTP file transfers, each of the 10 TCP flows sending a 1 GB file, with varying delay differential.

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 60 s difference in completion time, or about a 6% improvement. Fig. 8(c) shows that only HS-RR-TCP has timeouts at this level of loss. This is because the increase in *dupthresh* is causing timeouts. The loss is not high enough to cause timeouts with HS-TCP until 0.01 loss. At the same time, in Fig. 8(b) for loss of 0.001 we can see that it is still reducing the total number of 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. 8(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.

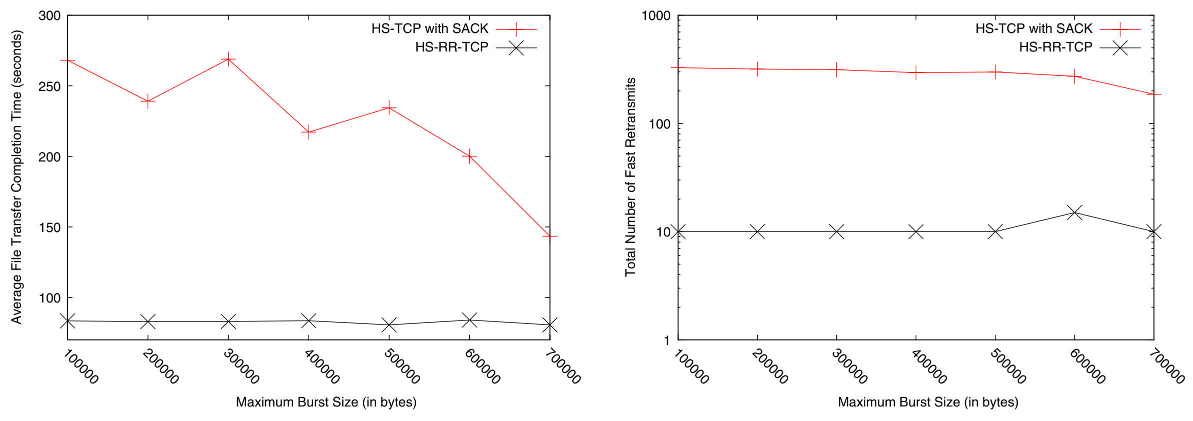
We also note that when we introduced loss in the access network, we obtained similar results (not shown due to space limitations).

5.2.4. *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. 9(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 OBS network is 1 Gbps (10 Mbps $\times$ 100 Mbps) 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. 9(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 100 kB with about a 4 s difference between the ρ_{max} values at opposite ends, while at the same time for HS-TCP-SACK there is an almost 190 s 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 79.5 s), so HS-RR-TCP



(a) Average file transfer completion time.

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

Fig. 7. Comparison of performance of FTP file transfers, each of the 10 flows sending a 1 GB file, with varying max burst sizes.

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.

5.2.5. *Performance with varying τ values*

The τ value is a parameter used for load-balancing that determines how often the path congestion is calculated. In our simulations we set this value to 500 ms. We ran simulations to vary this parameter. Fig. 12(a) shows the average completion time. With a smaller τ value, the path is changed more often leading to more path switches and more reorderings, which increases completion time for HS-TCP-SACK and also causes more fast retransmissions as seen in 12(b). HS-RR-TCP is unaffected by the changing τ values. At smaller τ values, HS-RR-TCP is over 400% faster than HS-TCP-SACK.

5.2.6. *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 s. 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. 10 compares the HS-RR-TCP parameters for history and FA ratio at 0.001 loss. From Fig. 10(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. 10(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. 10(b) also increases timeouts as seen in Fig. 10(d). We note that the single point for 10% FA ratio in Fig. 10(d) at 80 s history was caused by a small spike in *dupthresh* for some flows. This small spike was lost in the averaging of results, however, so it appears *dupthresh* remained constant in Fig. 10(c). We saw this behavior of decreased fast retransmission and increased timeouts previously in Section 5.2.3. 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.

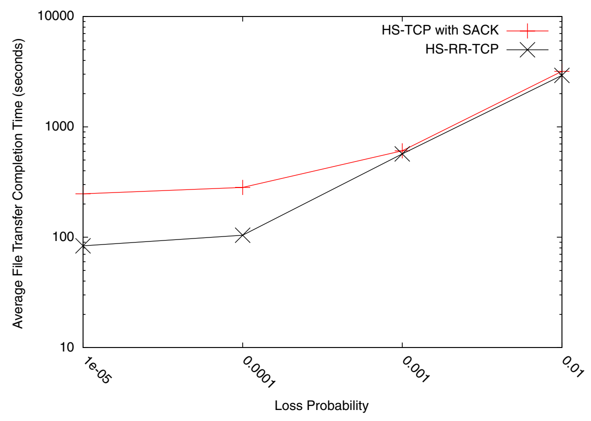
From these results, we conclude that a larger history, 80 s, and a higher FA ratio, 90%, are optimal values.

5.2.7. *Comparison of HS-RR-TCP and source ordering*

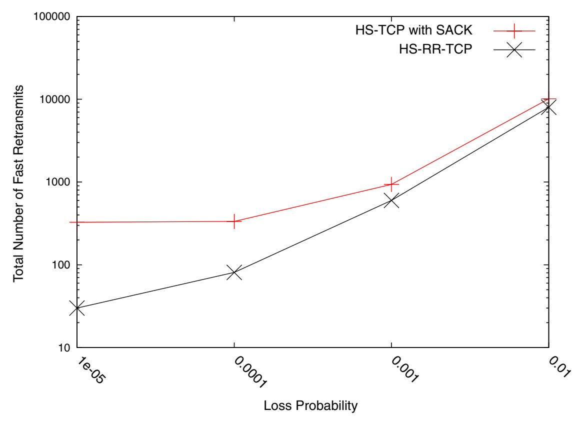
In this section we include source ordering in our results for varying delta and loss to compare the performance of HS-RR-TCP and source ordering. Fig. 11(a) shows that there is very little improvement using OBS-layer source ordering instead of HS-RR-TCP. The difference increases slightly as the delay differential increases, but at $\delta = 50$ ms there is only a difference of a few seconds, and path delay differentials greater than this are not realistic. From Fig. 11(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. 13. Fig. 13(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. 13(b) shows a 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.

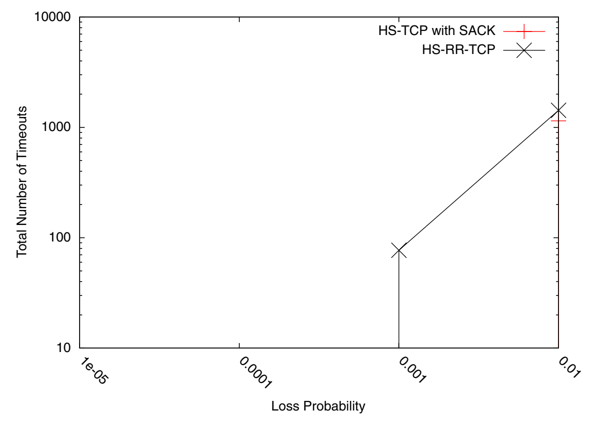
From these results, we can see that source ordering provides limited benefit compared to HS-RR-TCP. Source ordering requires adding electronic buffers at the ingress nodes while HS-RR-TCP is a transport layer mechanism



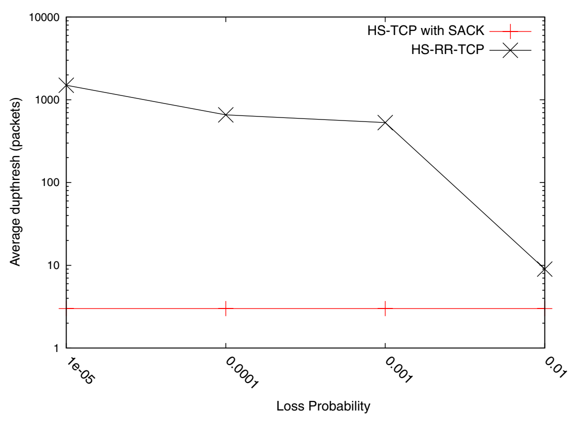
(a) Average file transfer completion time.



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

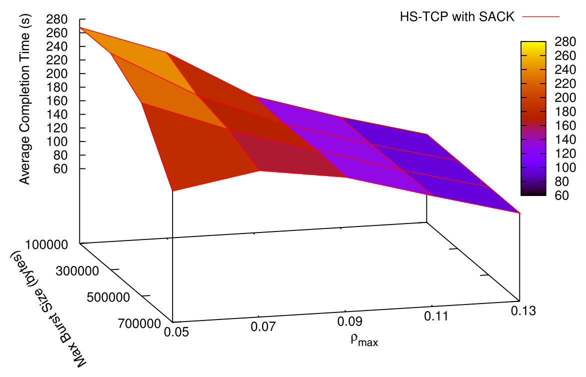


(c) Total number of timeouts across all flows.

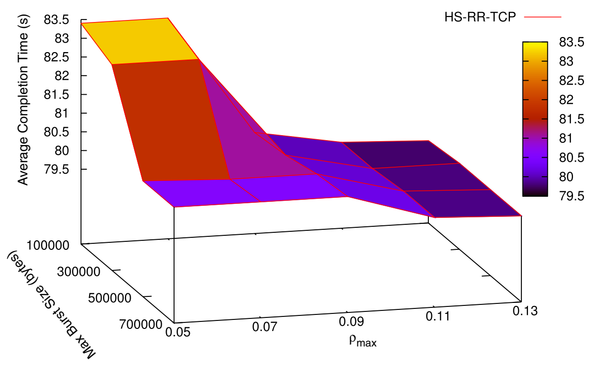


(d) Average *dupthresh* value across all flows.

Fig. 8. Comparison of performance of FTP file transfers, each of the 10 flows sending a 1 GB file, with random contentions.



(a) HS-TCP-SACK with varying $\rho_{\max}$ values.



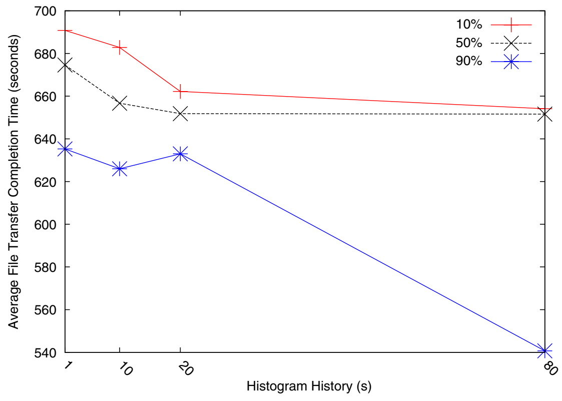
(b) HS-RR-TCP with varying $\rho_{\max}$ values.

Fig. 9. Comparison of HS-RR-TCP and HS-TCP-SACK for varying burst sizes and $\rho_{\max}$ values.

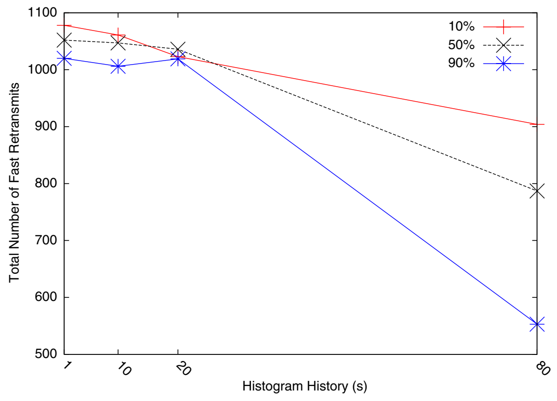
that modifies the SACK extension and does not increase asymptotic computation or storage complexity of SACK. Resolving reordering using HS-RR-TCP seems to be a better choice.

5.2.8. *HS-RR-TCP over NSFnet*

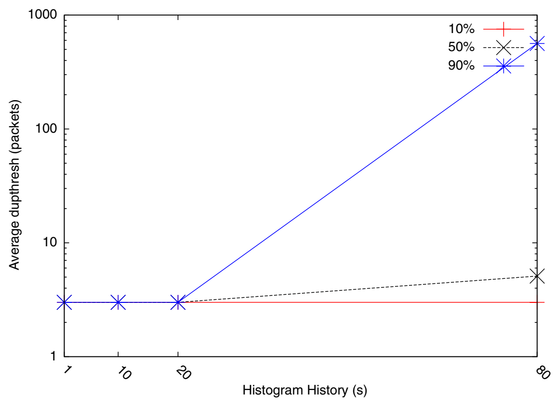
The previous simulations are run over a simple network in order to be able to precisely analyze the results and make conclusions about HS-RR-TCP's performance. In this subsection we briefly provide results obtained from running HS-RR-TCP over a more complicated topology with background traffic. We used the 14-node NSFnet (Fig. 14, distances in km) with background Pareto UDP traffic between all source-destination pairs. Each UDP source has an average send rate of 15 Mbps and a Hurst parameter of 0.75. Each link in the network has two data



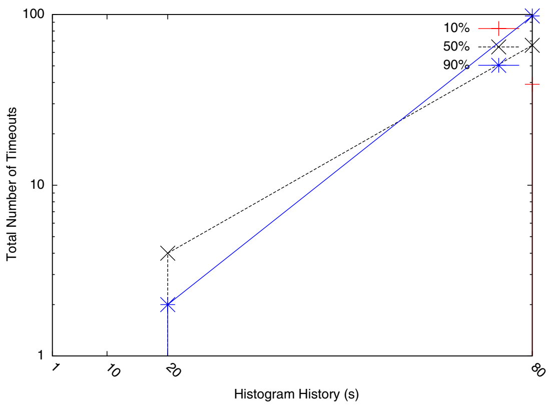
(a) Average file transfer completion time.



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

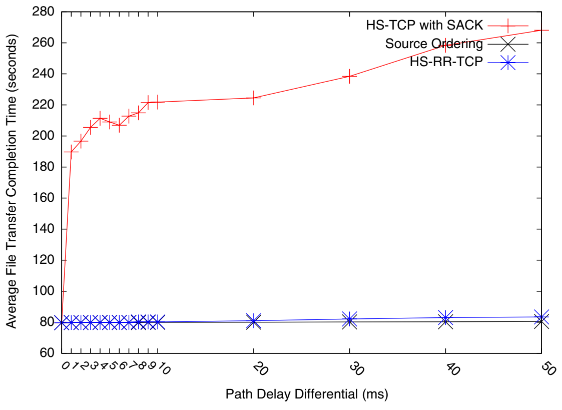


(c) Average *dupthresh* value.

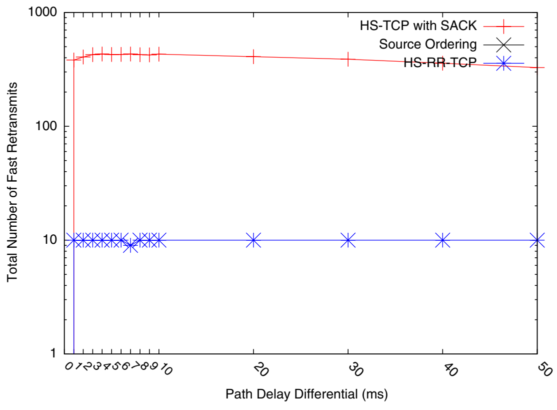


(d) Total number of timeouts across all flows.

Fig. 10. Comparing HS-RR-TCP performance with varying FA ratio and history at 0.001 loss.



(a) Average file transfer completion time.

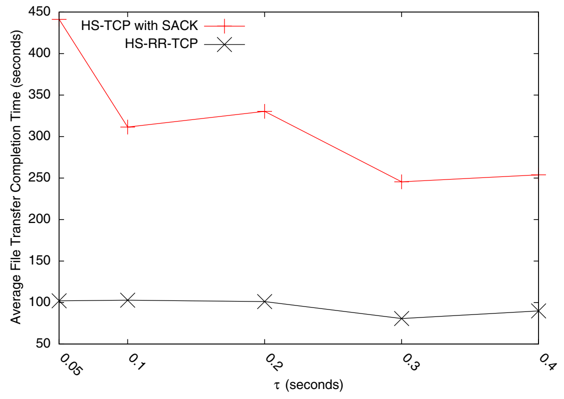


(b) Total number of fast retransmits across all flows.

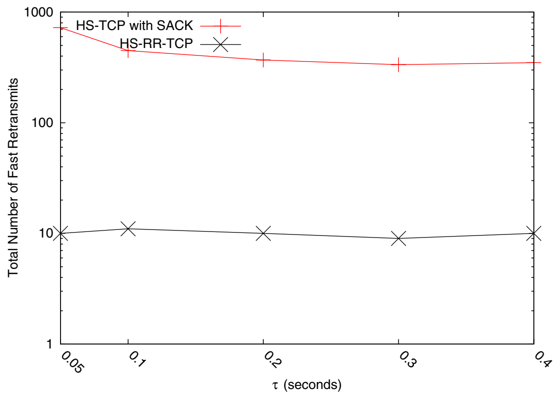
Fig. 11. Comparison of HS-RR-TCP and source ordering with varying δ.

channels operating at 1 Gbps each. The maximum burst size is 100 kB and the timer-threshold is 2 ms. We set $\tau = 300$ ms and $\rho = 5\%$. We ran simulations for three scenarios: no load-balancing with HS-TCP, load-balancing with HS-TCP, and load-balancing with HS-RR-TCP. For each scenario, there are eight source–destination pairs $\{(0, 4), (1, 5), (3, 9), (5, 7), (6, 13), (1, 12), (8, 0), (12, 8)\}$ with five TCP flows each sending 100 MB files for a total of 40 TCP flows. With load-balancing enabled, each of the eight pairs has a primary and alternate path. The delay differential between the two paths varies from 0.5 to 19 ms for different source–destination pairs.

We observe the following average completion times across all flows for the different scenarios: 177 s for no



(a) Average file transfer completion time.



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

Fig. 12. Comparison of performance of FTP file transfers, each of the 10 TCP flows sending a 1 GB file, with varying τ.

load-balancing with HS-TCP, 174 s with load-balancing and HS-TCP, and 142 s for load-balancing and HS-RR-TCP. We can see that introducing RR-TCP increases performance by an average of 20%.

6. Discussion

In this section we will briefly discuss our overall results. We have shown that HS-RR-TCP works well regardless of the delay differential (Fig. 6) and the amount of path switching that occurs (Fig. 9). We have shown that HS-RR-TCP always has performance at least as good as traditional HS-TCP regardless of the level of loss in the network (Fig. 8). We found good values for HS-RR-TCP specific parameters of FA and the histogram length (Fig. 10). We show HS-RR-TCP performs very similarly to the OBS-layer source ordering technique (Figs. 11 and 13). Lastly, we show that HS-RR-TCP performs well on more generic networks. We note there is a large performance difference between the simpler network and NSFnet. In the simpler scenarios (topology of Fig. 5) the large performance improvements resulted from there being very little or no TCP loss with burst reordering. We showed that with loss, the performance improvement is not as large. This is the case for NSFnet. We have actual TCP loss which limits the performance improvement gained by HS-RR-TCP. HS-RR-TCP performs the best with low TCP burst loss and burst reordering.

While we show that HS-RR-TCP performs well for load-balanced OBS networks, it is also applicable in other scenarios as well. We have shown HS-RR-TCP works well for a wide range of delay differentials and path switching frequency. This implies that any network that employs load balancing which may cause packet reordering can benefit from the use of HS-RR-TCP. This may include IP networks using MPLS traffic engineering functionality, or optical circuit switched networks using OSPF-TE, for example.

The main focus of this paper was to show that HS-RR-TCP works well for a range of load-balancing parameters. We did not discuss the selection of good $\rho_{\max}$ or τ values for overall network performance. For a detailed discussion of the selection of these parameters, we refer reads to our previous work [9]. The work in [9] does not consider TCP performance, but as we have shown HS-RR-TCP works well regardless of the load-balancing parameter settings, so the values recommended in our previous work can be recommended here as well.

7. Conclusion

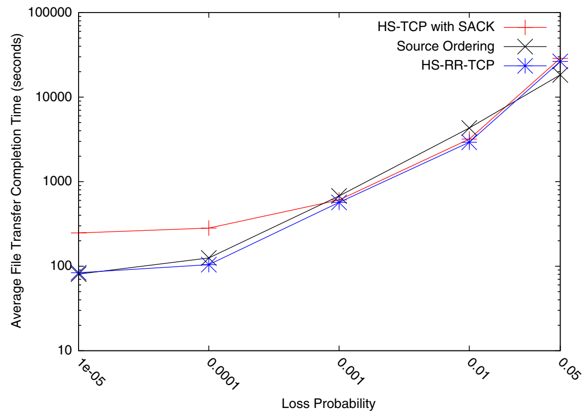
Load-balanced routing is a well-known loss minimization technique for OBS networks. Path switching causes persistent packet reordering that negatively affects the higher-layer TCP performance. We have presented an analytical end-to-end transfer delay model for TCP SACK and RR-TCP over a load-balanced OBS network. The model confirms TCP SACK's poor performance with persistent packet reordering. Based on extensive simulation results, we verified the analytical model and observed the benefits of RR-TCP mechanism under packet reordering in the load-balanced OBS network.

HS-RR-TCP showed significant improvement compared to HS-TCP with SACK, up to 300% under ideal scenarios and 20% in realistic scenarios with a complex network and background traffic. HS-RR-TCP successfully reduced the number of FFRs caused by packet reordering. The fundamental advantage of RR-TCP is that this solution does not require any changes to the OBS core, since it is a TCP layer modification, unlike an approach like source ordering that requires OBS layer modifications and also buffering. We also found that even very small delay differentials caused reordering which drastically impacted TCP's performance when HS-RR-TCP was not used.

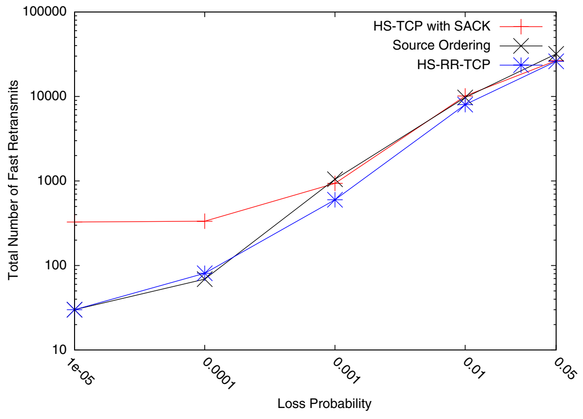
Appendix. Calculating *P*

The following determines the number of rounds, P, in slow start. Let T be the slow-start threshold, in packets.

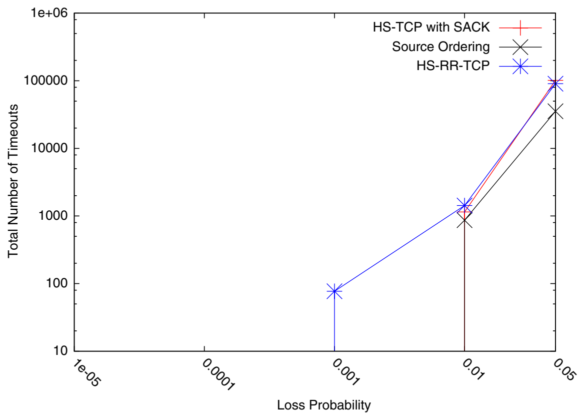
$$\begin{aligned}
P &= \min\{p : 2^0 + 2^1 + \cdots + 2^{p-1} \geq T\},\\
&= \min\{p : 2^p - 1 \geq T\},\\
&= \min\{p : p \geq \log_2(T+1)\},\\
&= \lceil \log_2(T+1) \rceil.
\end{aligned}$$



(a) Average file transfer completion time.



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



(c) Total number of timeouts across all flows.

Fig. 13. Comparison of HS-RR-TCP and source ordering with varying loss.

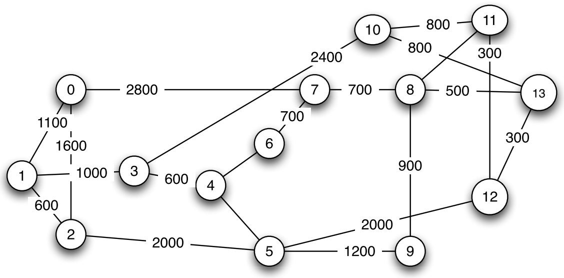


Fig. 14. NSFnet simulation topology.

References

[1] J. Jue, V. Vokkarane, Optical Burst Switched Networks, Springer, 2005.
[2] C. Qiao, M. Yoo, Optical burst switching (OBS)—a new paradigm for an optical Internet, Journal of High Speed Networks 8 (1) (1999) 69–84.
[3] I. Stoica, D. Karger, M.F. Kaashoek, H. Balakrishnan, Chord: a scalable peer-to-peer lookup protocol for Internet applications, in: Proceedings of ACM SIGCOMM, 2001.
[4] K.P. Gummadi, R.J. Dunn, S. Saroiu, S.D. Gribble, H.M. Levy, J. Zahorjan, Measurement, modeling, and analysis of a peer-to-peer file-sharing workload, in: ACM SIGMETRICS, 2003.
[5] I. Foster, C. Kesselman, S. Tuecke, The anatomy of the grid: enabling scalable virtual organizations, International Journal of High Performance Computing Applications 15 (2004) 200–222.
[6] A. Detti, M. Listanti, Impact of segments aggregation on TCP Reno flows in optical burst switching networks, in: Proceedings, IEEE INFOCOM, 2002.
[7] S. Yao, B. Mukherjee, S.J.B. Yoo, S. Dixit, A unified study of contention–resolution schemes in optical packet-switched networks, in: IEEE/OSA JLT, 2003.
[8] V.M. Vokkarane, J.P. Jue, Burst segmentation: an approach for reducing packet loss in optical burst switched networks, SPIE Optical Networks Magazine 4 (6) (2003) 81–89.
[9] G. Thodime, V.M. Vokkarane, J.P. Jue, Dynamic congestion-based load balanced routing in optical burst-switched networks, in: Proceedings, IEEE GLOBECOM, vol. 5, 2003, pp. 2694–2698.
[10] J. Li, M. Gurusamy, K. Chua, Load balancing using adaptive alternate routing in IP-over-WDM optical burst switching networks, in: SPIE OptiComm, vol. 5285, 2003, pp. 336–345.
[11] L. Yang, G. Rouskas, Adaptive path selection in OBS networks, IEEE/OSA Journal of Lightwave Technology 24 (8) (2006) 3002–3011.
[12] B. Komatireddy, N. Charbonneau, V.M. Vokkarane, Source-ordering for improved TCP performance over load-balanced optical burst-switched (OBS) networks, Photonic Network Communications 19 (1) (2010).
[13] M. Zhang, B. Karp, S. Floyd, L. Peterson, RR-TCP: a reordering-robust TCP with DSACK, in: Proceedings, IEEE International Conference on Networking Protocols, ICNP 2003, 2003, pp. 95–106.
[14] S. Floyd, Highspeed TCP for large congestion windows, RFC 3649. URL: http://www.ietf.org/rfc/rfc3649.txt (December 2003).
[15] J. Kurose, K. Ross, Computer Networking: A Top Down Approach Featuring the Internet, Addison-Wesley Longman, 2005.
[16] V.M. Vokkarane, Intermediate-node-initiation (INI): a generalized signaling framework for optical burst-switched networks, Elsevier Optical Switching and Networking (OSN) 4 (2007) 20–32.
[17] OBS-NS simulator: http://wine.icu.ac.kr/~obsns/index.php.