Noname manuscript No.
(will be inserted by the editor)

Multi-Layer Loss Recovery in TCP over Optical Burst-Switched (OBS) Networks

Rajesh R.C. Bikram · Neal Charbonneau · Vinod M. Vokkarane

Received: Monday, November 23, 2009; Revised: Monday, July 19, 2010

Abstract It is well-known that the bufferless nature of OBS networks causes random burst loss even at low traffic loads. When TCP is used over OBS, these random losses make the TCP sender decrease its congestion window even though the network may not be congested. This results in significant TCP throughput degradation. In this paper, we propose a multi-layer loss-recovery approach with ARQ and Snoop for OBS networks given that TCP is used at the transport layer. We evaluate the performance of Snoop and ARQ at the lower layer over a hybrid IP-OBS network. Based on the simulation results, the proposed multi-layer hybrid *ARQ + Snoop* approach outperforms all other approaches even at high loss probability. We developed an analytical model for end-to-end TCP throughput and verified with the simulation results.

Keywords: TCP, IP, ARQ, Snoop, and OBS.

1 Introduction

Optical burst switching (OBS) is a promising candidate to support the next-generation Internet. Optical burst switching provides a middle-ground between optical packet switching (OPS) and optical circuit switching (OCS). OBS is able to support bursty traffic that OCS cannot and is also technologically feasible in the near future since it does not require extremely fast switching as in OPS.

Packets arriving into the OBS network are assembled into bursts and subsequently transmitted through the network optically [1]. A burst header packet (BHP) is transmitted ahead of the associated burst in order to reserve the data channel and configure the core switches along the burst's route. The BHP carries information, such as source address, destination address, burst duration, and offset time of the associated burst. In just-enough-time (JET) signaling scheme [2], the transmission of the data burst follows an out-of-band BHP that is processed before the burst arrives at intermediate nodes. Most of the OBS signaling and reservation protocols are unidirectional in nature. They fail to inform the source about burst contentions along the route to the destination. Due to contentions at the core nodes, bursts will be dropped and the higher-layer protocol has to take responsibility to retransmit the lost data.

In recent years, TCP-based applications, such as Web (HTTP), email (SMTP), and peer-to-peer (P2P) file sharing, 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 for most of the TCP flavors, such as TCP SACK and TCP Reno is that they are carried on an electronic medium and packets experience queuing delays due to congestion at the IP router buffers. The bufferless nature of the OBS core network in addition to one-way JET signaling produces random burst losses even at low traffic loads. When TCP traffic traverses over OBS networks, the random burst loss may be falsely interpreted as network congestion by TCP.

There are three kinds of TCP flows, namely fast flows, medium flows, and slow flows [3]. For a fast flow, all the segments in a TCP source's sending window are assembled into a single outgoing burst. For a slow flow, at most one segment in a TCP source's sending window is included in any given outgoing burst. Thus, the loss of a burst will correspond to a single TCP segment being lost. For a medium flow, the number of segments for a TCP source included in a burst should be more than one and less than the sender's entire window size. If a fast flow based burst is dropped due

This work was supported in part of NSF Grant CNS-0626798

Department of Computer and Information Science, University of Massachusetts, Dartmouth, MA
E-mail: vvokkarane@ieee.org

to contention at a low traffic load, then the TCP sender times out, leading to false congestion detection, which is referred to as a *false timeout* (FTO) [4]. When the TCP sender detects this (false) congestion, it will trigger *slow start*, resulting in significantly reduced TCP throughput. Another example is when a random burst loss triggers TCP fast retransmission for medium flows or slow flows. The random burst loss in OBS will be interpreted as light congestion, leading to one or more TCP *false fast retransmissions* (FFR).

There are several contention resolution (or loss minimization) techniques, such as optical buffering [5], wavelength conversion [5], deflection routing [6], and segmentation [7]. Most loss minimization techniques are reactive in nature since they try to resolve the contention when it occurs. An alternative to loss minimization is to implement loss recovery techniques, such as cloning [8] and burst retransmission (ARQ) [9].

In this paper, we propose multi-layer data loss recovery for OBS networks. At the lower layer, we implement ARQ to minimize data loss due to random burst contentions. At the next higher layer, we implement Snoop to eliminate any FTOs/FFRs in it's network. Finally at the higher, TCP retransmit the lost packets using timeouts and fast retransmission mechanism. We implement the ARQ and Snoop layer to minimize redundancies and improve performance (throughput). Loss recovery is important in OBS to hide random burst losses from TCP that cause false congestion detection as discussed previously.This will be discussed in more detail in the following sections.

The remainder of the paper is organized as follows. Section 2 discusses background information, related work, and motivation. Section 3 describes how the multi-layer loss recovery is achieved over OBS and the components of our proposed approach. Section 4 describes the analytical modeling of proposed Snoop and ARQ over OBS. Section 5 discusses the numerical results, and Section 6 concludes the paper and discusses potential areas of future work.

2 Related Work and Motivation

While false congestion detection is an issue for TCP over OBS networks, it is also an issue in TCP over a wired-cum-wireless network. The main problem with TCP performance in networks that have both wired and wireless links is that packet losses that occurred due to bit-errors in the wireless network are mistakenly interpreted by the TCP sender as being due to network congestion. This causes the TCP sender to drop its transmission rate and often timeout, resulting in degraded throughput. Among the proposed solutions, Split-TCP shows significant improvement in the overall end-to-end throughput. In the Split-TCP approach for wireless networks, the base station works as a proxy between the sender and the mobile host. The base station ACKs received data from the TCP sender in the wired network and then relays the received data into the wireless network on behalf of the sender, allowing any loss in the wireless network to be hidden from the sender on the wired network. The performance increases by isolating the loss at wireless link from the sender.

The Split-TCP approach was recently modified to work on a hybrid IP-OBS network in [10] and was shown to provide significant improvement. In the Split-TCP approach over OBS, a single end-to-end TCP flow is divided into three independent TCP flows. One from the sending host to the optical ingress node (over the ingress IP-access), another TCP connection over the optical core, and the last connection from the optical egress node to the original destination host (over the egress IP-access). By splitting the connection, we can isolate burst contention losses over the OBS network from the IP-access networks. We can also implement different TCP flavors specific to each network segment that can boost end-to-end throughput. In the *N:1:N* Split-TCP approach, a persistent OBS-core TCP flow will be setup once and will continue until all the IP-access TCP flows have terminated. Although Split-TCP provided a significant performance improvement, the approach violates the end-to-end semantics of conventional TCP. Most Internet applications are sensitive to end-to-end TCP semantics.

The Snoop protocol [11] for wireless networks addresses the issues of TCP end-to-end semantics violation. Snoop works by deploying an agent at the base station and performing retransmissions of lost segments over the wireless portion based on duplicate TCP acknowledgments (which it also suppresses). Snoop does not acknowledge any data sent by the sender, so the end-to-end semantics are not violated. It simply tries to retransmit packets sent over the wireless network based on the receipt of duplicate ACKs before the TCP sender times out. This paper proposes Snoop for OBS networks.

Burst retransmission (ARQ) is a link layer loss recovery technique for OBS. In the burst retransmission approach, before sending a burst, the ingress buffers them in an electronic buffer. When a contention occurs in the OBS core, the core node sends an explicit ARQ message back to the OBS ingress. When the ARQ message is received, the ingress retransmits the burst from its buffer. It has been shown that ARQ improves performance compared to a regular OBS network but it behaves differently for different types of TCP flows. For a TCP fast flow, if a burst experiences contention and is successfully recovered by ARQ, there are no adverse side-effects. For medium flows, however, a burst contention may trigger fast retransmission even when OBS-layer retransmission is employed. In this case, packets from a given TCP flow may be spread across multiple bursts. Since the retransmitted burst incurs an extra retransmission delay, bursts that are sent after the contending burst may actually reach

the egress node prior to the retransmitted burst. The earlier arrival of these other bursts will result in the generation of duplicate ACKs, leading to the triggering of fast retransmission at the source. Once fast retransmission is triggered, the TCP sender will retransmit a lost packet and unnecessarily reduce its send rate.

We prefer the ARQ loss recovery approach since it is independent of TCP and does not violate end-to-end semantics like Split-TCP. However, we'd like to be able to overcome the issue of false fast retransmits caused by reordering of bursts. To do this, we propose using Snoop along with ARQ since Snoop is able to suppress duplicate acknowledgements. We discuss this approach in detail in the following sections.

3 Multi-layer Loss Recovery in OBS

In this section, we first discuss reliability over OBS in general, then discuss our proposed multi-layer approach. There are two main techniques to provide reliability in OBS networks. They can be classified as loss minimization and loss recovery mechanisms. Loss minimization involves either minimizing the probability of contentions or minimizing data loss after a contention. The former case is known as contention avoidance and is a proactive approach and the latter is a reactive approach known as contention resolution. An example of a contention avoidance approach is load-balanced routing [12]. Load-balancing tries to minimize the probability of contention by sending a burst on the least-congested path. An example of a contention resolution technique is deflection routing [6]. Deflection routing tries to reroute data after a contention by sending a burst to an alternate port.

Loss recovery involves either responding to explicit failure messages about a burst not being successfully transmitted or sending redundant information with each burst in order to recover from a loss. We can also divide loss recovery into proactive and reactive mechanisms. In proactive loss recovery, the OBS network assumes there will be contentions and therefore sends extra data (overhead) into the network to handle the contentions. An example of this would be burst cloning [8] where a redundant copy of each burst is sent into the network. If a original burst experiences contention, it is possible to recover using the clones. Reactive loss recovery mechanisms first assume that there will be no contentions in the network, but then responds to a failure if one does occur. Snoop and ARQ are both reactive loss recovery approaches. They send data into the network assuming it will be successfully received, but then respond to burst contentions. ARQ does so by retransmitting a burst upon receiving an ARQ request, while Snoop does so by handling duplicate ACKs. Generally, proactive approaches are used for delay-sensitive traffic and where loss probability is high while reactive approaches are used where loss probability is low and bandwidth is scarce.

Any of the above mechanisms can be combined to provide reliability in OBS. As previously mentioned, we evaluate the impact of combining two reactive loss recovery mechanisms: Snoop and ARQ. With multiple recovery mechanisms, we explore multi-layer reactive loss recovery over OBS.

In this paper, we evaluate the performance of three independent layers of loss recovery. The first being the transport layer, or TCP, which uses fast retransmit and timeouts as its loss recovery mechanisms. TCP is clearly independent of any lower layer recovery mechanisms. The other two layers of loss recovery are Snoop and ARQ, both working at the OBS layer, but at different granularity. The Snoop layer works on a packet-level and uses triple duplicates to determine when to attempt recovery, while ARQ works on a burst-level and uses explicit requests to attempt recovery. Both of these approaches are independent of one another but complement each other.

We are going to describe TCP, ARQ, and Snoop components of the proposed multi-layer loss recovery approach and then discuss how they are combined.

3.1 TCP

TCP congestion control mechanisms include slow start, congestion avoidance, fast retransmission, and fast recovery. If a TCP segment is lost, there are two types of loss indications: *time outs* (TO) and *triple duplicates* (TD). A TO loss is detected by a *retransmission timeout* (RTO) when an acknowledgment for a segment is not received within a certain period of time. TCP interprets a TO loss as a loss due to heavy network congestion; the TCP sender retransmits the lost segment and enters into a *slow start* phase. A TD loss is detected when a TCP sender receives three duplicate ACKs, which indicates that a packet is lost due to light network congestion; the TCP sender enters into *fast retransmission* and *fast recovery* without waiting for RTO.

3.2 ARQ

The basic idea of burst retransmission is to allow contending bursts to be retransmitted by the OBS layer rather than having higher-level protocols, such as TCP, recover lost data. In this mechanism, BHPs are sent out prior to data burst transmission in order to reserve resources. After an offset time, the burst is transmitted. The ingress node stores a copy of the transmitted burst for possible retransmissions. If the channel reservation fails at a core node due to burst contention, the core node will send an *automatic retransmission request*

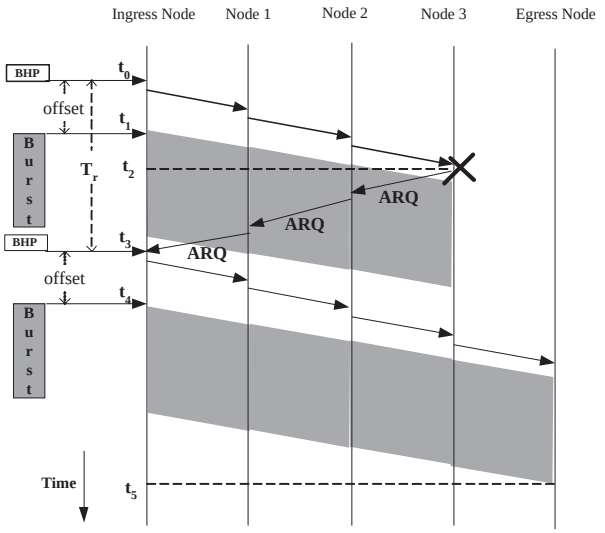


Fig. 1 ARQ approach.

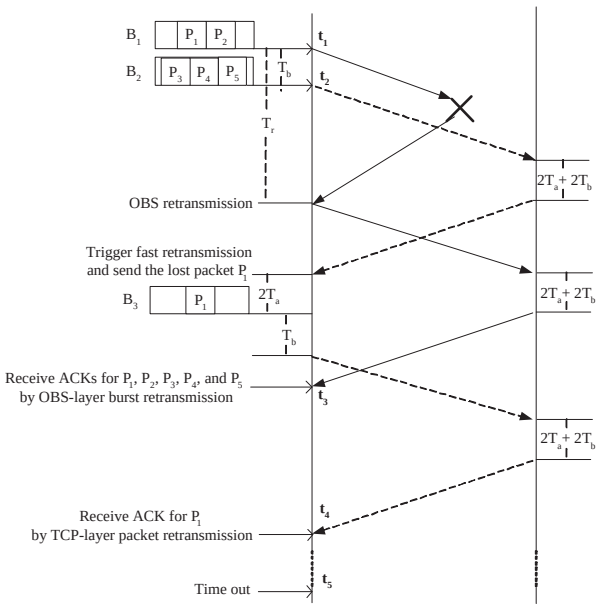


Fig. 2 Illustration of ARQ for medium flows.

(ARQ) to the ingress node in order to report the reservation failure. Upon receiving an ARQ, the ingress node retransmits a duplicate of the requested burst preceded by a corresponding BHP. If the network is lightly loaded, the retransmission scheme has a good chance of successfully retransmitting contending bursts. Hence, the retransmission mechanism improves the loss performance in an OBS network at the burst level. Since the contending burst can be retransmitted multiple times, if necessary prior to TCP timeout, ARQ can avoid FTOs for the above TCP layer. Retransmission may cause more fast retransmits when there are medium flows as retransmitted bursts may arrive out of order at the destination. If the network is heavily loaded, the retransmitted bursts have a lower probability of being successfully received. The ingress node can continue to attempt retransmission until the retransmission delay exceeds a threshold value, in which case the burst is dropped and no longer retransmitted when a contention occurs. We denote the delay incurred in the access network as T_a, the burst assembly and disassembly delay as T_b, the one-way propagation delay incurred in OBS network as T_p

We illustrate a bust retransmission scenario in Fig. 1. In this figure, the BHP is transmitted at time t_0, while the burst is duplicated and stored at the ingress node before being transmitted. The burst is transmitted at time t_1 after some offset time. At t_2, the burst reservation fails at Node 3, triggering Node 3 to send an ARQ back to the ingress node. The ingress node receives the ARQ at t_3, then sends a new BHP and retransmits a duplicate burst at t_4 after some offset time. Assuming the retransmission is successful, at t_5 the burst arrives at the egress node. A duplicate burst may be retransmitted multiple times until the burst successfully reaches the egress node or some threshold is reached.

As previously mentioned, ARQ can cause false fast retransmissions with medium TCP flows if bursts are reordered due to burst retransmission. Fig. 2 illustrates a scenario for a TCP medium flow with ARQ. Packets P_1, P_2, P_3, P_4, and P_5 belong to the same TCP sender's window. Packets P_1 and P_2 are assembled into burst B_1, and packets P_3, P_4, and P_5 are assembled into burst B_2. B_1 is dropped in the core while B_2 is successfully delivered. ARQ will retransmit B_1, but the ACKs from B_2 are duplicate *acks* and trigger a FFR at the sender even though B_1 is successfully retransmitted.

3.3 Snoop

We modify the standard Snoop approach for a hybrid IP-OBS network. Snoop performs local flow-level loss-recovery using the Snoop caches where packets transmitted by the TCP source are stored. Here, the Snoop agent resides on top of the OBS ingress. All the packets passing through the edge nodes are stored in the local Snoop caches per flows. Snoop uses these local caches to retransmit packets in the event of packet loss. Also, the Snoop agent suppresses duplicate acknowledgments received from the TCP receiver. The retransmission of lost packets by Snoop from its caches most of the time makes the TCP source unaware of random loss in the OBS network. This isolates the random loss in the OBS network from the TCP source allowing it to maintain a high send rate at most of the time.

When a packet arrives from the TCP source in-sequence and has a sequence number greater than the previous packet, Snoop caches the packet at the ingress and forwards the packet to the OBS burstification process. If an out-of-sequence

packet with sequence number less than the one already acknowledged is obtained, this is interpreted as a retransmitted packet from the TCP source. If this retransmitted packet was already received by the TCP receiver but the acknowledgement was lost, Snoop sends another acknowledgment packet back to the TCP source so that the TCP source does not trigger a timeout. Otherwise, the Snoop agent forwards the packet for burstification and caches this packet as being retransmitted by the TCP source.

Snoop uses two important functions for implementing its functionality, *snoop_data()* and *snoop_ack()* [11]. In the forward direction, the *snoop_data()* function processes and caches the packets before OBS burstification at the OBS ingress. In the reverse direction, the *snoop_ack()* function processes the acknowledgments received from the TCP receiver and performs packet retransmissions, if necessary. Whenever an *ack* is received from the OBS deburstification process, Snoop distinguishes it as either genuine *ack*, spurious *ack*, or *dupack*. If it is a genuine *ack*; an *ack* that is in order, Snoop clears its cache up to this sequence number and forwards the *ack* to the TCP source. If it is an *ack* with a sequence number smaller than the one previously received, it is a spurious *ack* and is discarded. If it is a *dupack* and the packet is not in the Snoop cache, it needs to be resent by the TCP source.

If the packet was a retransmitted packet from the TCP source, the *dupack* needs to be routed to the TCP source, because the TCP stack maintains state based on the number of duplicated acknowledgment it receives when it transmits a packet.

Consider a single packet loss scenario. If a specific packet is lost during transmission and if the successive packets in the flow successfully reach to TCP destination. The arrival of each successive packet in the window at the TCP destination causes a *dupack* to be generated for the lost packet. This can cause the TCP source to enter fast retransmission, there by reducing the send rate by half. In the presence of Snoop, the first *dupack* triggers retransmission of the lost packet at the OBS ingress. In addition the Snoop agent will suppress all the following *dupack* with the same sequence number leading to increase TCP throughput. In order to minimize the *dupack* as much as possible, the lost packet is retransmitted to burstification process as soon as the loss is detected at edge node where Snoop is implemented. For additional details about the implementation of Snoop refer [11].

3.4 TCP SACK Aware Snoop

From the illustration examples Fig. 3(a) we show the problem of TCP SACK with Snoop. Let us assume all the packets before sequence number 6 have been transmitted to the receiver successfully and that packets 7, 8, 9 and 10 have been dropped (these packets are part of burst which has been dropped in OBS network). Receiver sends a duplicate acknowledgment for packet 7 when packet 11 is received by it. This duplicate acknowledgment when received by Snoop makes it retransmit packet 7 from its local cache and suppress the *dupacks* for 7. In the meantime, packet 12 generates another duplicate acknowledgment for packet 7. When packet 7 is received by the receiver, it sends acknowledgment packet 8. All the acknowledgments received by the Snoop are actually SACK blocks asking the sender to send packets 7, 8 , 9, and 10 in one RTT. With the introduction of Snoop between sender and receiver only retransmitted packet 7 and all duplicate *acks* (which are SACK blocks) are dropped. The SACK sender is thus not able to retransmit all the packets in one RTT and the basic SACK mechanism is thus disrupted. If a SACK block such as that having sequence number 8 reaches the SACK source, a retransmission of all the packets 8, 9, and 10 takes place. These packets upon reaching Snoop are dropped as Snoop already has them in its cache and considers them unnecessary retransmissions from the sender. This once again leads to bandwidth wastage.

In order to analyze the problem of unnecessary retransmissions of the SACK sender and enable Snoop to retransmit all the missing packets in one RTT, the TCP SACK Aware Snoop algorithm is proposed [9]. As the name suggests, this algorithm helps Snoop differentiate between an ordinary *ack* and an acknowledgment containing a SACK block.

From the illustration Fig. 3(b), we show how SACK aware Snoop handles the problem generated by the TCP sack over Snoop. Assume that packets 7, 8, 9 and 10 are lost and that packets 7, 8, and 9 are present in the Snoop cache where as 10 is not, the SACK block received due to the duplicate acknowledgment for 7 indicates the SACK sender to retransmit these missing packets. When TCP SACK Aware Snoop receives this packet and determines that it is a SACK block, it immediately retransmits the packets 7, 8, and 9, as 13 is the largest continuous sequence number that is available in its cache, and drops the duplicate acknowledgment. Upon reception of these packets, the receiver generates an acknowledgment requesting 10 and also indicating the reception of packets 7, 8 and 9. This acknowledgment is forwarded to the sender so that it does not timeout for any of the packets 7, 8 or 9. The sender retransmits packet 10 after the reception of three duplicate acknowledgments. Packet 10 is now stored in the Snoop cache and forwarded to the receiver. This helps in reducing the sender retransmissions of packets and also enables retransmission of multiple packets in one RTT thus maintaining a good flow of packets.

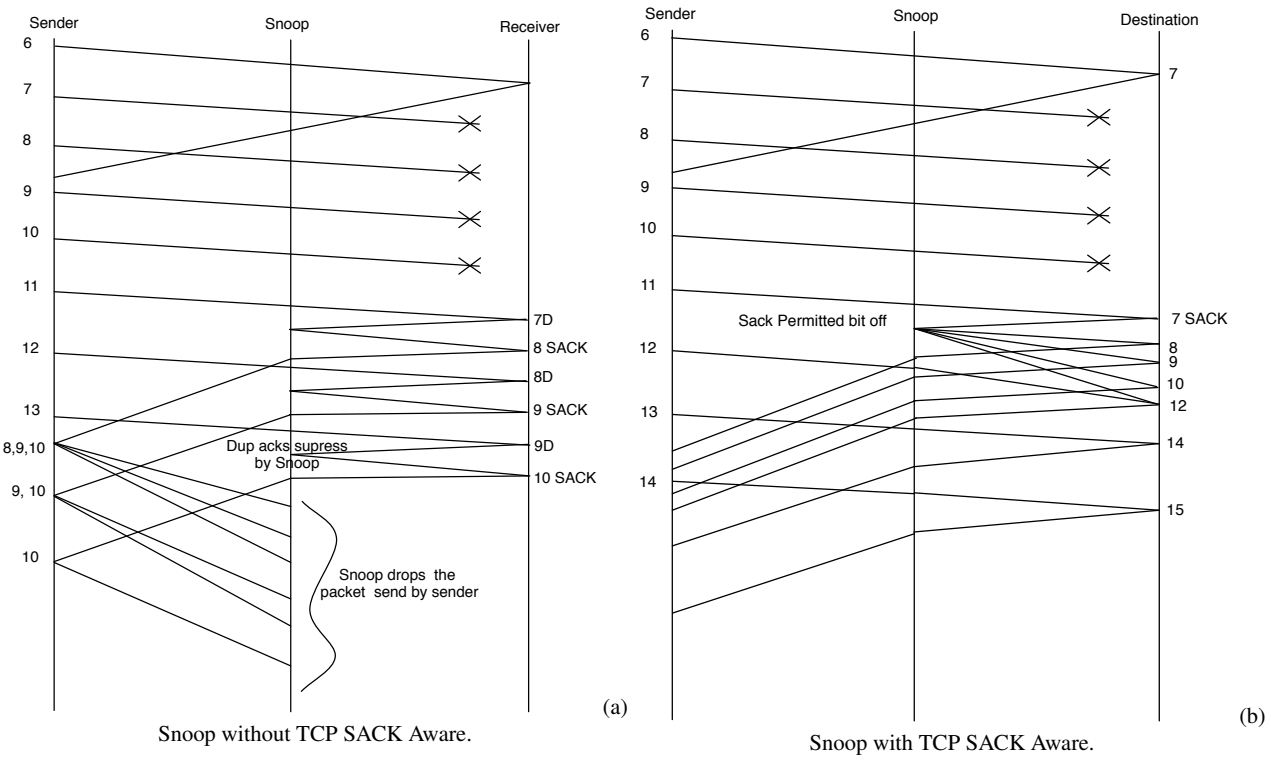


Fig. 3 TCP SACK performance with and with out TCP SACK Aware Snoop

3.5 Hybrid (ARQ + Snoop)

As we have shown, ARQ causes FFRs with medium TCP flows. Snoop can hide the FFRs caused by ARQ from the sender since it suppresses *dupacks*. We propose a hybrid approach that implements both Snoop and ARQ. Both of them perform loss recovery, where Snoop performs recovery at the packet level and ARQ at the burst level. Snoop is triggered by the receipt of a duplicate acknowledgment while burst retransmission is triggered by explicit ARQ messages. In most cases, ARQ prevents timeouts by retransmitting the bursts, but ARQ can trigger FFRs (as shown in Fig. 2) due to the out-of-order delivery of bursts. By using Snoop on top of ARQ, we can prevent the FFRs.

Let us consider a loss scenario with *ARQ+Snoop* implemented at the OBS ingress. Each new packet sent by the TCP source is stored in the Snoop cache (handled by $snoop\_data()$) before being forwarded to the OBS burstification process. A burst and its corresponding BHP are created during the burstification process. After the burst is created, a copy of the burst is stored in a retransmission buffer. In the event of a burst loss, the OBS ingress will receive an ARQ. A duplicate burst is retransmitted from the retransmission buffer. If the TCP destination receives packets out of order, it sends duplicate acknowledgements back to the TCP source. These duplicate acknowledgements are received as a burst on the reverse path. During deburstifications, the $snoop\_ack()$ will detect the duplicate acknowledgements, and using the Snoop cache, the $snoop\_data()$ will retransmit the missing packet and suppress the duplicate acknowledgement.

The burst loss event can lead to two scenarios. In the first scenario, ARQ successfully retransmits the lost burst and Snoop suppresses all of the duplicate acknowledgements. In second scenario, ARQ is unable to recover the lost burst. In this case, Snoop has to individually retransmit all the lost packets in addition to suppressing duplicate acknowledgements.

If there is no loss in the OBS core, there is no additional control overhead. Each acknowledgement received in the correct order removes the corresponding data packets from the Snoop cache.

3.6 Multi-layer loss recovery architecture

Fig. 4 depicts the multi-layer loss recovery architecture for OBS networks. The TCP end-to-end flow is from *A-B-C-D*. Snoop operates over the flow segment *B-C-D*, while ARQ operates only over the flow segment *B-C*. If both Snoop and ARQ are unable to recover the lost packets, then TCP at the transport layer recovers it.

Two order of loss recovery can be achieved using Snoop and ARQ. First order reliability is by ARQ retransmission. ARQ retransmission fails only if either ARQ exceeds the delay constraints time, ρ, as described in [9] or number of

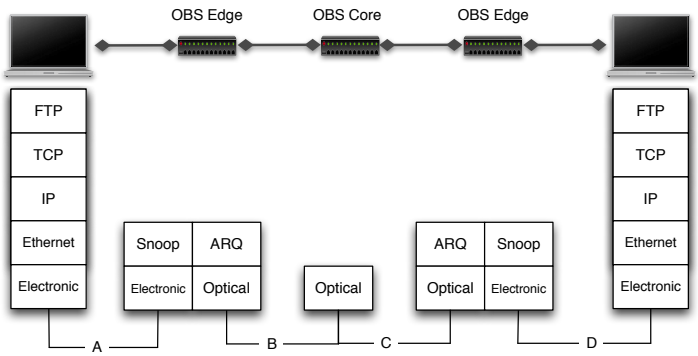


Fig. 4 Multi-layer loss recovery architecture.

retransmission exceeds for lost burst. Snoop usually retransmits the packets if it receives duplicate acknowledgment. To control the delay and overflow of the packets at Snoop, Snoop tries to retransmit the packets for specified threshold numbers and gives up by notifying congestion in network if it exceeds the threshold of packet retransmission.

4 Performance Analysis

In this section, we develop an analytical model for evaluating the end-to-end burst loss probability for OBS networks with two layers of retransmission; burst level and packet level. Burst level retransmission is done by ARQ whereas packet level retransmission is done by Snoop. We developed an analytical model for evaluating TCP throughput when TCP is implemented over an OBS network with ARQ and Snoop.

We analyze TCP throughput for a TCP fast flow and a TCP medium flow. For the TCP fast flow, all three TCP flavors have the same behavior. For the TCP medium flow, since TCP SACK performs the best, we only analyze TCP SACK throughput. In TCP fast flows with ARQ, successful retransmission in the OBS network do not produce any false fast retransmissions (FFRs) and false timeouts (FTOs) when burst is lost in OBS network. In this scenario, Snoop explicitly does not have any effects as Snoop does not get any duplicate acknowledgements during fast flows. ARQ failure to retransmit the lost burst in OBS for fast flows generates FTOs. In this scenario, Snoop does not have direct impact on the overall TCP performance as we know Snoop gets triggered only by duplicate acknowledgements. During medium flows, ARQ may generate FFRs when the burst is lost in the OBS network. Snoop controls this situation by suppressing the duplicate acknowledgment and retransmits the packets of the lost burst to burst assembly process. If ARQ get succeed to retransmit the lost burst, Snoop retransmits unnecessary packet, which will be a research for future work. TCP medium flows for ARQ with Snoop behaves as fast flows for ARQ only. With Snoop FFRs get suppressed and sender usually never gets duplicated acknowledgements as in fast flows. Our analytical modeling is base on [9] paper.

In this section, we analyze the TCP throughput over an OBS network with ARQ and Snoop. In the existing analysis of TCP throughput over an OBS network without burst retransmission [3],[9], burst loss probability is equal to burst contention probability. When OBS retransmission is employed, a burst is only considered lost if it experiences contention and it is not successfully retransmitted. Thus, the burst loss probability differs from the burst contention probability. In the analysis, we must take into account both burst loss probability and burst contention probability which includes the packet retransmission scenario due to Snoop.

As defined in [9], a TCP sending *round* refers to the period during which all packets in the sending window are sent and the first *acks* for one of the packets in the sending window is received. We assume that the time needed to send all the packets in the sending window is less than RTT. Hence, the duration of a round is equal to RTT. We also assume that the number of packets that are acknowledged by a received ACK is one. Furthermore we assume that Snoop has sufficient cache to accommodate each packet received from the senders.

We introduce the following notation for a TCP flow:

p_c: burst contention probability.
p_d: burst dropping probability.
B: TCP throughput.
W_m: TCP maximum window size (in packets).
Z^{TO}: duration of a sequence of TOs.
H: number of TCP segments sent in Z^{TO}.

4.1 TCP Medium Flow

Our analysis of a TCP medium flow is similar to that in [3]. However, in our analysis for the case with OBS retransmission, the successfully retransmitted bursts are treated differently from the bursts that do not experience any contention. The retransmitted bursts suffer from an extra retransmission delay, which has a negative effect on the TCP throughput.

Medium flow triggers TD events, but these TD events are suppressed by Snoop. The medium flow imitates the TCP fast flow behavior, so we are using TCP fast flow modeling instead of medium flow modeling.

Since a TCP fast flow does not trigger TD, multiple successful sending rounds are only followed with one or multiple lossy rounds. Therefore, as in [3], a given time out period includes a sequence of successful rounds and a sequence of lossy rounds. In this time out period, let X be the number of successful rounds, Y be the number of segments sent before the first lossy rounds, and A be the duration of the sequence of successful rounds. We can then calculate the TCP

throughput as given below:

$$B^f = \frac{E[Y] + E[H]}{E[A] + E[Z^{TO}]}. \quad (1)$$

The sequence of successful rounds consists of a portion of rounds in which the burst does not experience contention and a portion of rounds in which the burst experiences contention, but is successfully retransmitted. Hence, we obtain the probability of a successful round in which a burst experiences contention but is successfully retransmitted as

$$p_{sr} = \frac{p_c - p_d}{1 - p_d}. \quad (2)$$

The probability of a successful round in which there is no burst contention can be calculated as

$$p_{nc} = \frac{1 - p_c}{1 - p_d}. \quad (3)$$

We assume that each retransmission of a burst takes an average time of T_p, where T_p is one way propagation delay incurred in OBS networks. Then, the average number of retransmissions for a retransmitted burst, given that the burst needs to be retransmitted at least once and the retransmission is successful, is

$$E[r_b] = \sum_{i=1}^{\lfloor \delta/T_p \rfloor - 1} i p_c^{i-1}(1 - p_c) + \lfloor \frac{\delta}{T_p} \rfloor p_c^{(\lfloor \frac{\delta}{T_p} \rfloor - 1)}. \quad (4)$$

where $E[r_b]$ is average RTT due to burst retransmission.

We assume that each retransmission of a packet takes an average time of T_p (assuming that packets are already in the Snoop cache). Then, the average number of retransmission packets, given that the packet need to be retransmitted at least once and the retransmission is successful, is

$$E[r_s] = \sum_{i=1}^{N-1} i p_c^{i-1}(1 - p_c) + N p_c^{(N-1)}. \quad (5)$$

where, N is total number of packet retransmission and $E[r_s]$ is average number of RTT due to packet retransmission.

Hence, the average round trip time experienced by a successful burst and packet retransmission by ARQ and Snoop, respectively is

$$RTT_r = RTT + E[r_b]T_p + E[r_s]T_p, \quad (6)$$

$$= RTT + T_p(E[r_b] + E[r_s]). \quad (7)$$

We then obtain $E[A]$ as

$$E[A] = p_{sr} E[X] RTT_r + p_{nc} E[X] RTT. \quad (8)$$

Based on the equations (14), (16), (18), and (28) in [3], we have

$$E[Z^{TO}] = RTO \frac{f(p_d)}{1 - p_d}, \quad (9)$$

where, $f(p_d) = 1 + p_d + 2p_d^2 + 4p_d^3 + 8p_d^4 + 16p_d^5 + 32p_d^6$.

$$E[H] = \frac{p_d}{1 - p_d}, \quad (10)$$

$$E[X] = \frac{1 - p_d}{p_d}, \quad (11)$$

and

$$E[Y] = \begin{cases} \frac{1}{p_d^2} & p_d > \frac{1}{W_m} \\ \frac{W_m}{p_d} & otherwise. \end{cases} \quad (12)$$

Since only burst losses result in TOs for a fast flow, the burst loss probability in an OBS network with burst retransmission is applied in the above equations.

By substituting equations (8), (9), (10), and (12) into (15), we have

$$B^f = \frac{p_d^3 - p_d + 1}{p_d[(1-p_d)(p_c - p_d)RTT_r + (1-p_d)(1-p_c)RTT + p_d f(p_d) RTO]}, \quad (13)$$

when $p_d > \frac{1}{W_m}$, and

$$B^f = \frac{p_d^2 + W_m - W_m p_d}{(1-p_d)(p_c - p_d)RTT_r + (1-p_d)(1-p_c)RTT + p_d f(p_d) RTO}, \quad (14)$$

when $p_d \leq \frac{1}{W_m}$.

4.2 TCP Fast Flow

Our analysis of a TCP fast flow is similar to that in [3]. Snoop comes in effect with the recipient of duplicate acknowledgments. With TCP fast flow, burst loss in OBS produces timeouts and Snoop isolates itself for TCP fast flows. Throughput modeling of Snoop with ARQ is the same as just ARQ modeling. We are using ARQ modeling for TCP fast flow from the paper [9].

We can calculate TCP throughput as

$$B^f = \frac{E[Y] + E[H]}{E[A] + E[Z^{TO}]}. \quad (15)$$

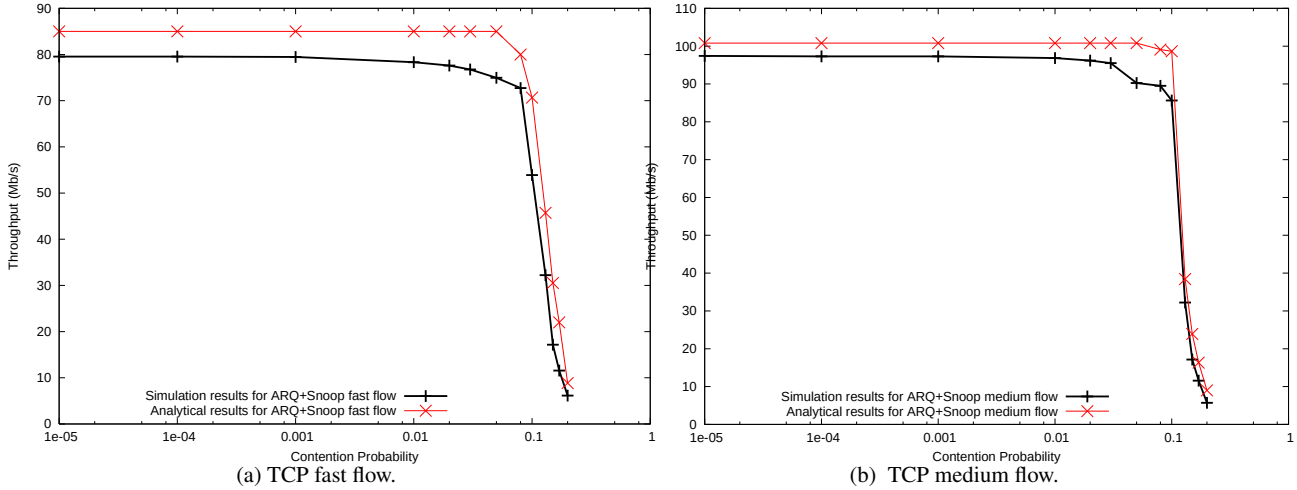
We use equations (6), (7), (8), and (10) in [9] to obtain

$$B^f = \frac{p_d^3 - p_d + 1}{p_d[(1-p_d)(p_c - p_d)RTT_r + (1-p_d)(1-p_c)RTT + p_d f(p_d) RTO]}, \quad (16)$$

when $p_d > \frac{1}{W_m}$, and

$$B^f = \frac{p_d^2 + W_m - W_m p_d}{(1-p_d)(p_c - p_d)RTT_r + (1-p_d)(1-p_c)RTT + p_d f(p_d) RTO}, \quad (17)$$

when $p_d \leq \frac{1}{W_m}$.



(a) TCP fast flow. (b) TCP medium flow.

Fig. 5 Analytical and Simulation results for TCP fast and medium flows

5 Performance Analysis

5.1 Analytical Validation

We developed analytical model to verify our simulation results. The analytical model evaluates the performance of TCP over an OBS network with ARQ and Snoop. Fig. 5(a) and (b) compares the analytical and simulation results for TCP SACK through in OBS network with independent Snoop and ARQ module.

We simulate a network shown in Fig. 6(a) with one TCP flow that shares a common link. Each flow has the following assumptions, $W_m = 10000$ and $T_p = 40\ ms$. We assume access bandwidth has $B_a = 100\ Mbps$ and $T_b = 10\ ms$, where T_b is burst assembly and disassembly delay, for TCP medium flows and high burst size and less sender window for TCP fast flows. Packets are dropped if Snoop retransmission failed after one retransmission and burst are dropped if ARQ exceeds $\rho = 2T_b + 2T_p$). We see that simulation results matches with analytical results.

5.2 Simulation Results

The topology used for this simulation is shown in Fig. 6(a). There are a total of two edge nodes and three cores nodes denoted by *E* and *C*, respectively. There is an IP access network connected to both the ingress and egress OBS nodes. The access networks have four nodes with one TCP sender on each node. We run available bit-rate UDP traffic source at $50\ Mb/s$ between $C1$ to $C3$ for all the simulation plots. An FTP traffic generator is used to send a $1\ GB$ file over each of the five flows, which are TCP SACK flows. TCP's advertised window remains constant throughout the simulation.

Each link has *16* data channels with *1 Gbps* bandwidth on each channel. The edge nodes use a mixed timer-threshold burst assembly mechanism with a timer of *10 ms* and a maximum burst size of *50 KB*. Burst contention is simulated by randomly dropping bursts at core node $C2$.

In our simulations, Snoop attempts one retransmission per packet and the Snoop cache size can go up to *9 MB* for each TCP flow. Snoop Buffer management is out of scope for this paper. ARQ uses a maximum of three retransmission attempts to handle burst loss. We compare the performance of regular OBS, OBS with Snoop, OBS with ARQ, and OBS with *ARQ+Snoop* over varying loss probabilities in the core.

In Fig. 6(b) we compare the average file transfer completion time. The figure shows that *ARQ+Snoop* significantly improves TCP performance. The completion time stays almost constant for all loss probabilities. At 1% contention probability there is over an order of magnitude improvement over *regular OBS* and at 10% almost three orders of magnitude. There is also some improvement in completion time between *ARQ* and *ARQ+Snoop* as shown in Fig. 6(c), ARQ causes a large number of FFRs due to reordering of bursts. As the contention probability increases, ARQ causes more fast retransmissions. From Fig. 6(c) and (d) we observe that Snoop alone handles FFRs well while ARQ alone handles FTOs well. Therefore, the combination of the two, *ARQ+Snoop*, provides significant performance improvement.

We can see that without any loss recovery, regular OBS performs very poorly. As from the Fig. 6(e) that the average congestion window is small enough for the entire window to fit inside a single burst. As a result, when a burst is dropped the TCP sender will always enter timeout instead of fast retransmission. By itself, Snoop does not provide large performance improvement because it cannot prevent time-

outs even though it handles fast retransmissions as shown in Fig. 6(c) and (d).

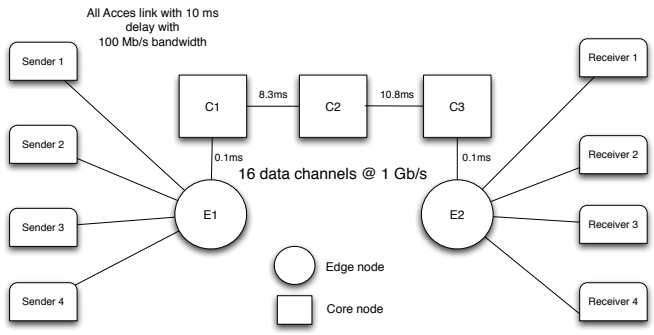
In Fig. 6(f) we compare the average per flow throughput. Again, from the graphs we see *ARQ+Snoop*performs better than other schemes as the contention probability increases. There is significant improvement compared to ARQ alone because of the FFRs caused by ARQ. Snoop handles fast retransmission by suppressing duplicate acknowledgments but cannot handle FTOs. The two approaches complement each other to handle both FTOs and FFRs.

6 Conclusion

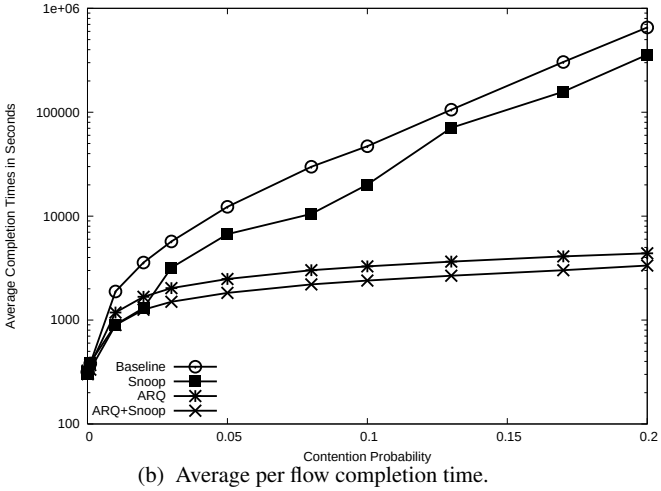
In this paper we have evaluated multi-layer loss recovery mechanisms, including *Snoop*, *ARQ*, and *ARQ+Snoop* for TCP over OBS networks. We have shown that the independent *ARQ+Snoop* approach over OBS significantly improves overall TCP performance. Comparing the average end-to-end TCP flows completion time, *ARQ+Snoop* is up to two orders of magnitude faster than *regular OBS* while also providing performance improvements over *ARQ*, and *Snoop*.

References

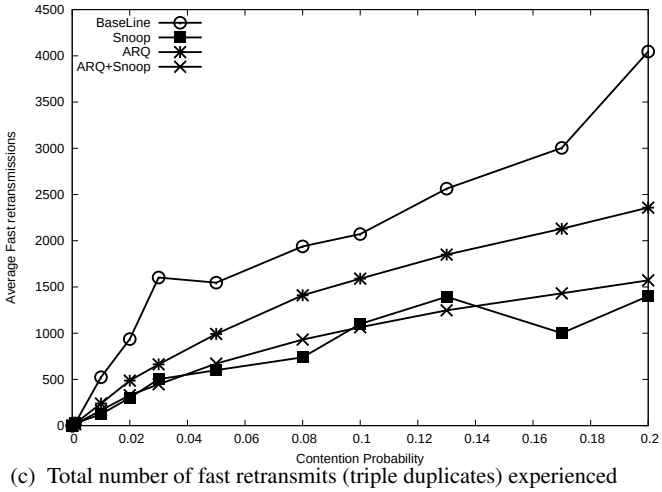
1. J.P. Jue and V.M. Vokkarane, *Optical Burst Switched Networks*, Springer, 2005.
2. C. Qiao and M. Yoo, "Optical burst switching (OBS) - a new paradigm for an optical Internet," *JHSN*, vol. 8, no. 1, pp. 69–84, Jan. 1999.
3. A. Detti et. al., "Impact of segments aggregation on TCP Reno flows in optical burst switching networks," in *INFOCOM*, 2002, vol. 3, pp. 1803–1812.
4. X. Yu et. al., "TCP implementations and false time out detection in OBS networks," in *Proceedings, IEEE INFOCOM*, Mar. 2004, vol. 2, pp. 774 – 784.
5. I. Chlamtac et al., "CORD: Contention resolution by delay lines," *IEEE JSAC*, vol. 14, no. 5, pp. 1014–1029, Jun. 1996.
6. S. Yao et. al., "A unified study of contention-resolution schemes in optical packet-switched networks," *IEEE/OSA Journal of Lightwave Technology*, vol. 21, no. 3, pp. 672 – 683, 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. X. Huang et. al., "Burst cloning: A proactive scheme to reduce data loss in optical burst-switched networks," in *Proceedings, IEEE International Conference on Communications (ICC)*, May 2005, vol. 3, pp. 1673 – 1677.
9. Q. Zhang et. al., "Analysis of TCP over optical burst-switched networks with burst retransmission," *IEEE Globecom*, vol. 4, pp. 1978–1983, Nov. 2005.
10. R.R.C. Bikram and V.M. Vokkarane, "TCP over optical burst switching (OBS): To split or not to split?," *IEEE/OSA Journal of Lightwave Technology (JLT)*, vol. 27, no. 23, pp. 5208–5219, Dec. 2009.
11. H. Balakrishnan, V.N. Padmanabhan, S. Seshan, and R.H. Katz, "A comparison of mechanisms for improving TCP performance over wireless links," *IEEE/ACM Transactions on Networking*, vol. 5, no. 6, pp. 756 –769, Dec. 1997.
12. B. Komatireddy, N. Charbonneau, and V.M. Vokkarane, "Source-ordering for improved TCP performance over load-balanced optical burst-switched (OBS) networks," *Springer Photonic Network Communications*, vol. 19, no. 1, pp. 1–8, Feb. 2010.



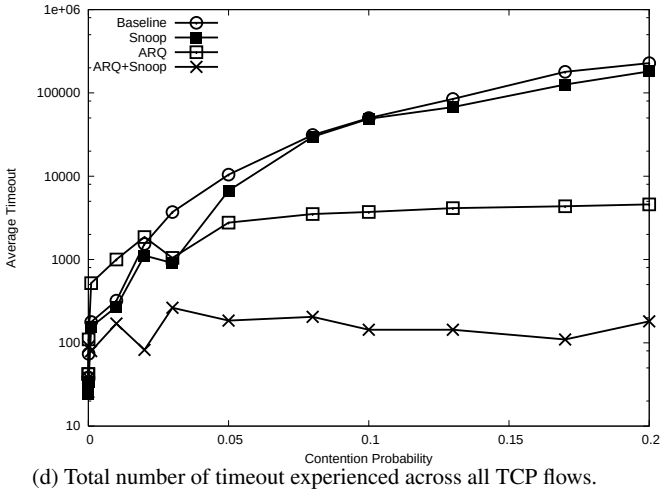
(a) Simulation network topology.



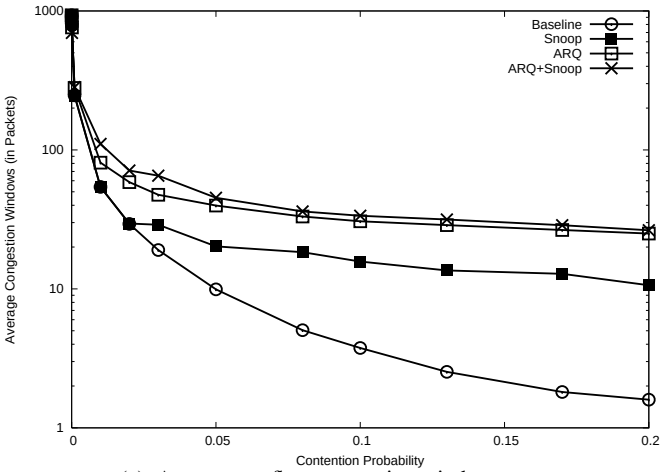
(b) Average per flow completion time.



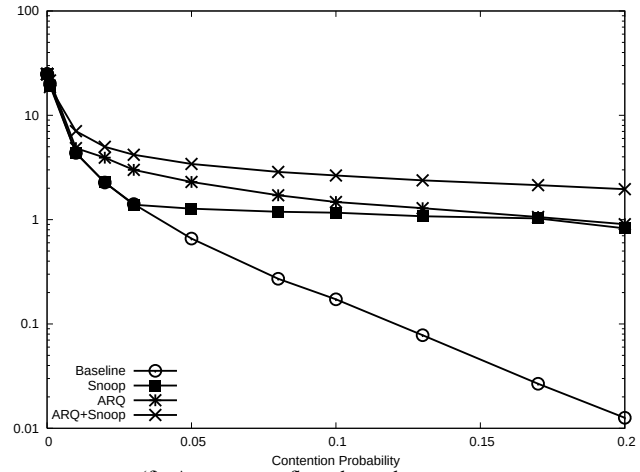
(c) Total number of fast retransmits (triple duplicates) experienced across all TCP flows.



(d) Total number of timeout experienced across all TCP flows.



(e) Average per flow congestion windows.



(f) Average per flow throughput.

Fig. 6 TCP performance with different contention probability.