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

Rajesh RC Bikram, Neal Charbonneau, and Vinod M. Vokkarane
Department of Computer and Information Science, University of Massachusetts, Dartmouth, MA
E-mail: {rrc, u_ncharbonne, vvokkarane}@umassd.edu

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 coordinated multi-layer loss-recovery approach for TCP over OBS networks using Snoop and ARQ. We developed an analytical model for end-to-end TCP throughput using the hybrid approach over OBS and verified the results using simulations. We evaluate the performance of independent and coordinated Snoop and ARQ over an OBS network. Based on the numerical results, the proposed coordinated multi-layer Snoop and ARQ approach outperforms all other approaches at all traffic loads.** [1]

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

I. Introduction

Optical burst switching (OBS) is a promising candidate to support the next-generation Internet. 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.

In recent years, TCP-based applications, such as web (HTTP), email (SMTP), cloud computing, 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 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 to a random contention at low traffic loads , 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]. Instead of loss minimization, loss recovery techniques, such as cloning [8] and burst retransmission (ARQ) [9] can be used.

In this paper, we propose a coordinated multi-layer loss recovery approach for OBS networks. At the lowest 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. Finally, TCP is used to retransmit any lost packets using timeouts and fast retransmissions. We implement simple coordination between Snoop and ARQ layers to minimize redundancy and improve performance.

The remainder of the paper is organized as follows: Section II discusses related work. Section III describes how multi-layer loss recovery is implemented over an OBS network. Section IV describes the analytical model of the proposed coordinated approach. Section V discusses the numerical results, and Section VI concludes the paper.

II. Related work

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 window and often timeout, resulting in degraded throughput. Among the proposed solutions, Split-TCP [10] shows significant improvement in the overall end-to-end throughput. However, Split-TCP violates the end-to-end semantics of TCP.

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. In this paper, we incorporate Snoop for providing packet-level loss recovery over OBS networks.

Burst retransmission (ARQ) is a link-layer loss recovery approach for OBS. In the burst retransmission approach, before transmitting a burst, the ingress buffers them in an

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

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 [9]. 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 retransmissions 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.

The ARQ loss recovery approach is independent of TCP and does not violate end-to-end semantics like Split-TCP. In order to overcome the issue of FFRs caused by reordering of bursts in ARQ, we propose using Snoop along with ARQ. Snoop is able to suppress duplicate acknowledgements (*dupacks*) generated by ARQ. We discuss this approach in detail in the following sections.

III. 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.

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. Reactive loss recovery mechanisms first assume that there will be no contentions in the network, but then responds to a contention if one does occur. Snoop and ARQ are both reactive loss recovery approaches. ARQ recovers by retransmitting a burst upon receiving an ARQ request, while Snoop recovers 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.

We can further classify multi-layer loss recovery mechanisms into independent and coordinated multi-layer loss recovery. In the coordinated approach the layers know about each other and communicate to provide reliability. While with independent multi-layer recovery, each layer does not need to know about the other. Both independent and coordinated approaches are used to analyze Snoop and ARQ over OBS network.

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 at the packet-level and uses triple duplicates to determine when to attempt recovery, while ARQ works at the 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 how independent and coordinated multi-layer loss recovery using Snoop and ARQ work over OBS networks with an illustration example.

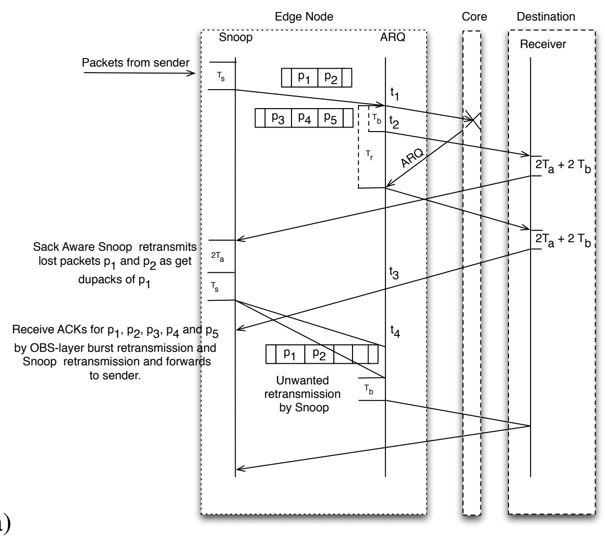
A. Independent Snoop and ARQ

ARQ causes FFRs with TCP medium flows as we mentioned earlier. Snoop can hide the FFRs caused by ARQ from the TCP sender since it can suppress *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 *dupacks* while burst retransmission is triggered by explicit ARQ messages. In most cases, ARQ prevents timeouts by retransmitting lost bursts, but ARQ can trigger FFRs due to the out-of-order delivery of bursts. By using Snoop on top of ARQ, we can prevent these FFRs.

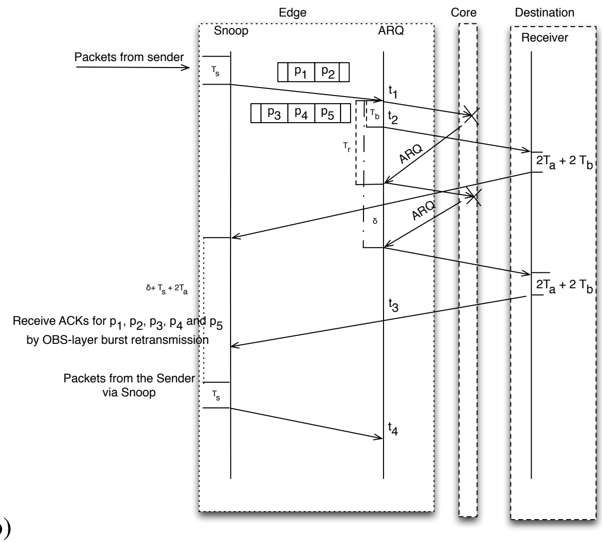
Let us consider a loss scenario with the Snoop and ARQ approach implemented at the OBS ingress. Each new packet sent by the TCP source is stored in the Snoop cache before being forwarded to the OBS burstification process. A burst and its corresponding BHP are created during the burstification process. 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 *dupacks* back to the TCP source. These *dupacks* are received as a burst on the reverse path. During deburstification, the Snoop will detect the *dupacks*, and using the Snoop cache, the Snoop 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.


Edge Node
Core
Destination
Snoop
ARQ
Receiver
Packets from sender
Sack Aware Snoop retransmits lost packets p1 and p2 as get dupacks of p1
Receive ACKs for p1, p2, p3, p4 and p5 by OBS-layer burst retransmission and Snoop retransmission and forwards to sender.
Unwanted retransmission by Snoop
ARQ
2Ta + 2 Tb
t1
t2
t3
t4

(a)


Edge
Core
Destination
Snoop
ARQ
Receiver
Packets from sender
Receive ACKs for p1, p2, p3, p4 and p5 by OBS-layer burst retransmission
Packets from the Sender via Snoop
ARQ
2Ta + 2 Tb
t1
t2
t3
t4

(b)

Fig. 1. (a) Independent and (b) Coordinated Snoop and ARQ.

B. Coordinated Snoop and ARQ

In this section, we develop the mechanism to coordinate between Snoop and ARQ. We illustrate in Fig. 1 the problems with Snoop and ARQ if they work independent of each other. If Snoop somehow knows about the ARQ status, then it can try to prevent unwanted packet retransmission from its Snoop cache. Snoop uses locally saved retransmission timers for each packet, and the ARQ information for that packet to decide whether to delay packets or not. The delay constraint, ρ, information from the ARQ is passed to the Snoop. The delay constraint specifies how long ARQ keeps a burst before removing it from its buffer and therefore giving up on retransmission. More details about calculation of ρ is described in paper [9]. The coordinated scenario is more important when ARQ retransmission is successful when one of the sequence of burst gets dropped in the core node. 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, and the delay incurred by Snoop cache as T_s. We will describe the problem with an illustrative example.

Fig. 1(a) illustrates a scenario for a TCP medium flow where Snoop and ARQ work independently. Since Snoop and ARQ are implemented at the edge nodes, there is negligible propagation delay between them. Let 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. Lets suppose B_1 is dropped in the core while B_2 is successfully delivered. ARQ will retransmit B_1 in its delay constraint time, ρ, while in the mean time Snoop receives duplicate *acks* for B_1 triggering Snoop to retransmit the lost packets, P_1 and P_2, in same RTT (as Snoop is sack-aware), even though B_1 is successfully retransmitted by ARQ. These unwanted packet retransmission from Snoop causes waste of bandwidth.

To overcome the unwanted retransmission problem by Snoop as in Fig. 1(a), ARQ sends the ρ information to the Snoop so that it can delay its retransmission timer. By delaying the retransmission timer, Snoop receives the acknowledgment for B_1 before the Snoop retransmits the lost packets through its local cache. From the illustration Fig. 1(b) Snoop delays the retransmission timer by $\rho + 2T_a$ so that Snoop can always guarantee $t_3 < t_4$. This means that Snoop received the *acks* for P_1, P_2, P_3, P_4, and P_5 before Snoop retransmitted the packets P_1 and P_2 (as Snoop is Sack aware). In this way, we minimize the unwanted packet retransmission when ARQ retransmission is successful as illustrated in Fig. 1(b). Coordination between Snoop and ARQ always adds to the overall end-to-end delays but these added delays for Snoop are always much less than overall end-to-end TCP delays.

IV. 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 where as packet level retransmission is done Snoop. We developed an analytical model for evaluating TCP throughput when TCP is implemented over an OBS network with Snoop and ARQ.

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,too 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 burst is lost in the OBS network. Snoop control this situation by suppressing the duplicate acknowledgment and retransmits the packets of lost burst to burst assembly process. 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.

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 ($b = 1$). 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:	# of TCP segments sent in Z^{TO}.

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:

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

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

$$p_{sr} = \frac{p_c - p_d}{1 - p_d}. \tag{2}$$

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

$$p_{nc} = \frac{1 - p_c}{1 - p_d}. \tag{3}$$

We assume that each retransmission of a burst takes an average time of T_p. 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

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

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

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

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 Snoop and ARQ, respectively, is given by

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

where RTT_r is round trip time when Snoop and ARQ works independently. The RTT for coordinated Snoop and ARQ is given by,

$$RTT_r = RTT_r + \rho + 2 * T_p. \tag{7}$$

We then obtain $E[A]$ as

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

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

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

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

$$E[H] = \frac{p_d}{1 - p_d}, \tag{10}$$

$$E[X] = \frac{1 - p_d}{p_d}, \tag{11}$$

and

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

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 (1), we have

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

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

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

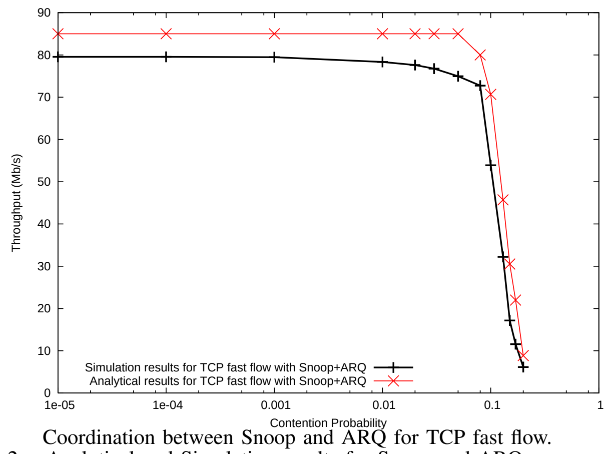
when $p_d \leq \frac{1}{W_m}$.

2) TCP fast flow: Our analysis of a TCP fast flow is similar to that in [3]. Snoop comes into effect with the receipt of *dupacks*. With TCP fast flow, burst loss in OBS leads to only TCP timeouts and Snoop isolates itself for TCP fast flows. Throughput modeling of Snoop with ARQ is the same as just ARQ modeling. Coordination between Snoop and ARQ does not have any meaning as Snoop does not receive any duplicate acknowledgment as most of the time burst lost in network results TCP timeouts. We are using ARQ modeling for TCP fast flow from the paper [9].

We can calculate TCP throughput as

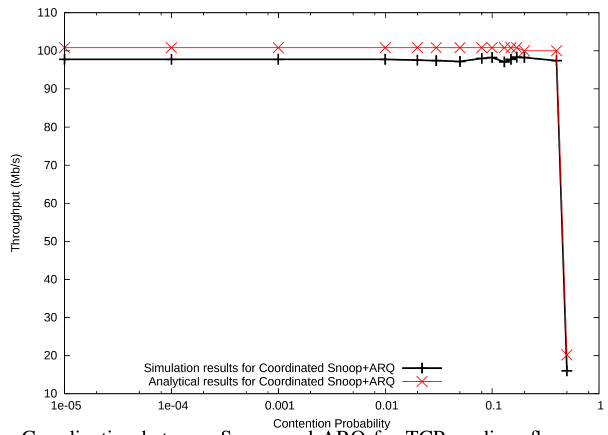
$$B^f = \frac{E[Y] + E[H]}{E[A] + E[Z^{TO}]}. \tag{15}$$

(a)



Coordination between Snoop and ARQ for TCP fast flow.

(b)



Coordination between Snoop and ARQ for TCP medium flow.

Fig. 2. Analytical and Simulation results for Snoop and ARQ.

We use equation (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 $p_d \leq \frac{1}{W_m}$.

V. Numerical analysis

We developed analytical model to verify our simulation results. The analytical model evaluates the performance of TCP over an OBS network with Snoop and ARQ. Fig. 2(a) and (b) compares the analytical and simulation results for TCP Sack in OBS network with coordinated Snoop and ARQ for TCP fast flow and medium flow respectively. We simulate a network [12] with 1 TCP flow. Each flow has the following assumptions, $W_m = 10000$ packets and $T_p = 40ms$. We assume access bandwidth has $B_a = 100Mbps$ and $T_b = 10ms$ for TCP medium flow and $W_m = 1000$ for TCP fast flow. 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.

The topology used for this simulation is shown in Fig. 3(a). There are a total of 2 edge nodes and 3 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. The nodes in the left access network send data to the nodes in the right access network. An FTP traffic generator is used to send a $1GB$ file over each of the flows, which are TCP Sack flows. TCP's advertised window remains constant throughout the simulation. Each link has *16* data channels with *1Gbps* bandwidth on each channel. The edge nodes use a mixed timer-threshold burst assembly mechanism with a timer of *10ms* and a maximum burst size of *50KB*. Burst contention is simulated by randomly dropping bursts at core node *C2*.

In Fig. 3(f) we compare the total burst received by OBS egress with and without coordination between Snoop and ARQ. With the increase of the contention probability loss the total burst received by destination increases with the coordination as it suppress the unwanted retransmission packets from the Snoop and retransmits the new packets to ARQ.

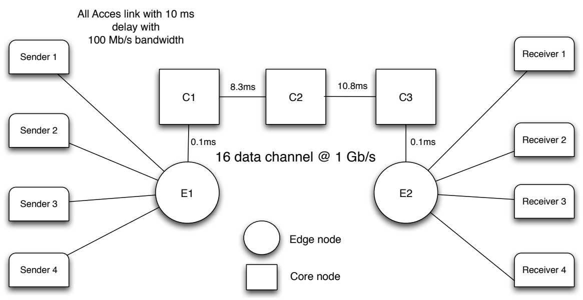
In our simulations, Snoop attempts one retransmission per packet and the Snoop cache size goes up to *9Mb* for each TCP. ARQ uses a maximum of three retransmission attempts to handle burst loss. We compare regular OBS, OBS with Snoop, OBS with ARQ, and OBS with and without coordinated Snoop and ARQ over varying loss probabilities in the core.

In Fig. 3(b) we compare the average file transfer completion time. The graph shows that coordinated Snoop and ARQ 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 a significant improvement in completion time between ARQ and Snoop+ARQ with or without coordination between them. Fig. 3(c) shows that this is because ARQ causes a large number of FFRs due to reordering of bursts. As the contention probability increases, ARQ causes more fast retransmissions. From Fig. 3(c) and (d) we observe that Snoop alone handles FFRs well while ARQ alone handles FTOs well. Therefore, the combination of the two, *Snoop+ARQ*, provides significant performance improvement.

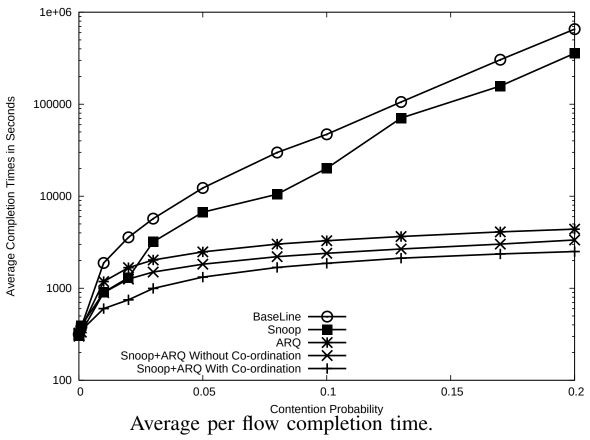
We can see that without any loss recovery, regular OBS performs very poorly. We can see from Fig. 3(e) that the average congestion window is small enough for the entire window to fit inside a single burst. As a result, when a burst is dropped the TCP sender will always enter timeout instead of fast retransmission. By itself, Snoop does not provide large performance improvement because it cannot prevent timeouts even though it handles fast retransmissions as shown in Fig. 3(c) and (d). Coordinated *Snoop+ARQ* 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. These congestion window values translate directly to throughput obtained as well.

VI. Conclusion

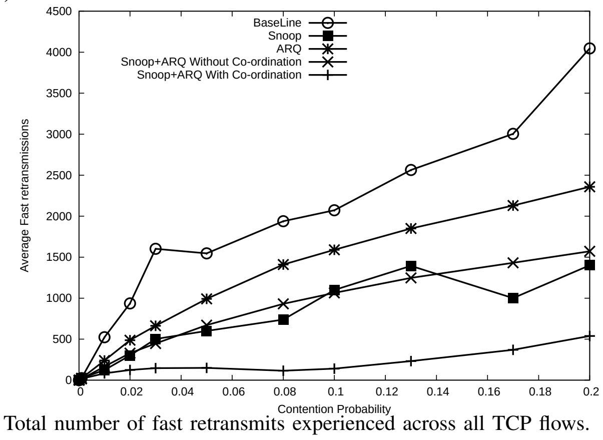
In this paper we have evaluated multi-layer loss recovery mechanisms, including Snoop, ARQ, and *Snoop+ARQ* coordinated and independent for TCP over OBS networks. We have shown that the coordinated *Snoop+ARQ* approach over OBS

(a)


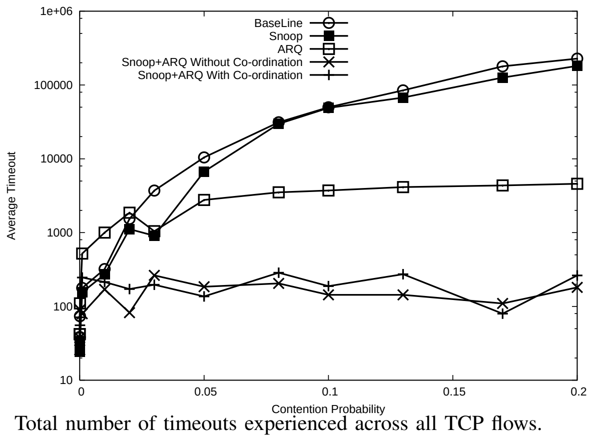
Simulation network topology.

(b)


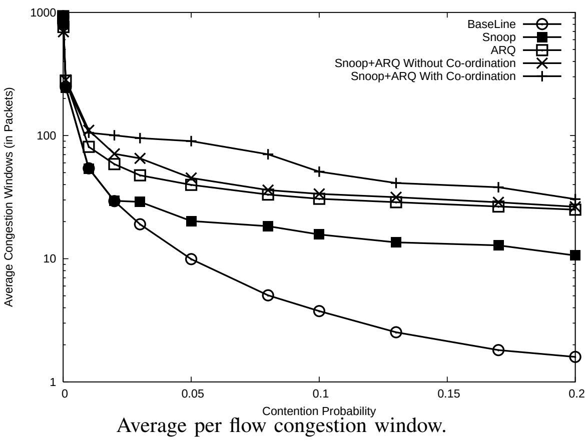
Average per flow completion time.

(c)


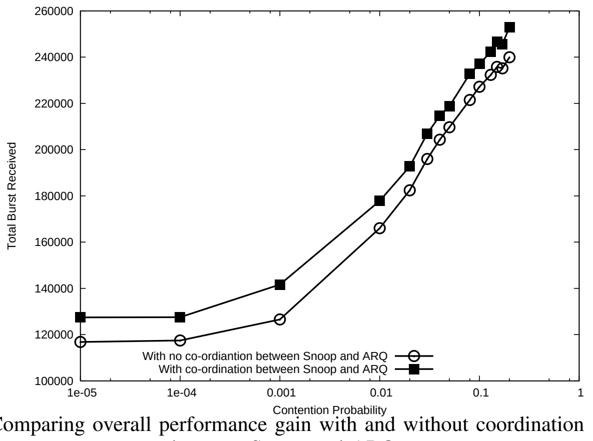
Total number of fast retransmits experienced across all TCP flows.

(d)


Total number of timeouts experienced across all TCP flows.

(e)


Average per flow congestion window.

(f)


Comparing overall performance gain with and without coordination between Snoop and ARQ.

Fig. 3. TCP performance with different contention probability.

significantly improves overall TCP performance. Comparing the average end-to-end TCP flows completion time, coordinated *Snoop+ARQ* is up to two orders of magnitude faster than regular OBS while also providing performance improvements over ARQ, independent *Snoop+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.
[4] X. Yu et. al., "TCP implementations and false time out detection in OBS networks," in *Proceedings, IEEE INFOCOM*, Mar. 2004.
[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," in *IEEE/OSA JLT*, 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.
[9] Q. Zhang et. al., "Analysis of TCP over optical burst-switched networks with burst retransmission," *IEEE Globecom*, Nov. 2005.
[10] S. Kopparty et. al., "Split TCP for mobile ad hoc networks," in *Proceedings, IEEE GLOBECOM*, Nov. 2002, vol. 1, pp. 138–142 vol.1.
[11] H. Balakrishnan et. al., "Improving TCP/IP performance over wireless networks," in *Proceedings, 1st ACM Conf. on Mobile Computing and Networking*, Nov. 1995.
[12] "OBS-NS simulator: http://dawn.cs.umbc.edu/owns/," .