Performance evaluation of TCP over OBS considering background traffic

Óscar González de Dios, Ignacio de Miguel, Víctor López

***Abstract*—Optical burst switching is a promising alternative for next generation optical networks. On the other hand, TCP is expected to continue as the prevailing transport protocol. Therefore, TCP over OBS needs to be studied in depth. Previous work on this topic has usually considered the transmission of a single TCP flow through the OBS network. However, a more pragmatic scenario is that where the burstifier is fed by multiple data flows. In this paper we study, by means of simulation, the performance of TCP Reno and SACK over OBS networks in that scenario. We compare the results with those obtained for the single flow situation, in terms of the number of segments carried per burst and goodput, showing significant differences in behavior and performance.**

***Index Terms*—TCP, OBS, multiple traffic sources**

I. Introduction

The spectacular growth of Internet traffic has fostered research in optical networking, due to the huge amount of bandwidth offered by the optical fiber. A number of switching paradigms have been proposed with the aim of designing these networks. Wavelength-routed optical networks are based on optical circuit switching [1]. It is a relatively mature technology, but usually leads to inefficient use of bandwidth. An alternative to improve bandwidth utilization as well as to enhance adaptability to traffic conditions are optical packet-switched networks. However, optical packet switching is still in its early stages and lacks of maturity [2]. An intermediate and promising option is Optical Burst Switching (OBS) [3]. OBS combines advantages from both optical circuit and packet switching, managing bandwidth efficiently, and being a feasible technology in the medium term. For these reasons there is a significant interest in this technology, which has led to the proposal of architectural variations, to studies about performance, and even to the implementation of OBS testbeds.

In OBS networks, packets —usually IP datagrams—, arrive at the edge nodes, where they are classified in electronic buffers according to their destination and class of service, and are grouped into bursts (Figure 1). These bursts are transmitted through the core optical network towards the destination edge node, where the burst is disassembled into IP packets. Prior to the transmission of a burst, a burst control packet (BCP) is sent. The BCP usually contains information about the destination edge node, the length of the burst, the offset time between the BCP and the burst, and the channel (typically a wavelength) in which the burst will be sent. Since intermediate nodes receive the BCP some time before the arrival of the burst, they have enough time to determine the next hop for the control packet (and for the burst), to select the wavelength to use, and to configure the switching matrix of the node, so that all resources are ready exactly when the burst arrives.

One of the key issues in OBS is the method employed to aggregate the packets into bursts. This process is usually known as burstification. There are several assembly algorithms [3-6]. One method commonly used, and the one that we will use in this study, is the timeout-based assembly method. When a packet arrives at a buffer (and if a burst formation process has not started yet), a new burst begins to be assembled, and a timer is activated. The following packets arriving at the buffer are added to the burst until the timer expires. Other algorithms are based on burst size. These methods aggregate packets until the size of the burst reaches a size threshold. There are also hybrid approaches, which combine the utilization of a timer and a size threshold, and adaptive algorithms, which dynamically modify the timer and/or the size threshold according to traffic conditions [6].

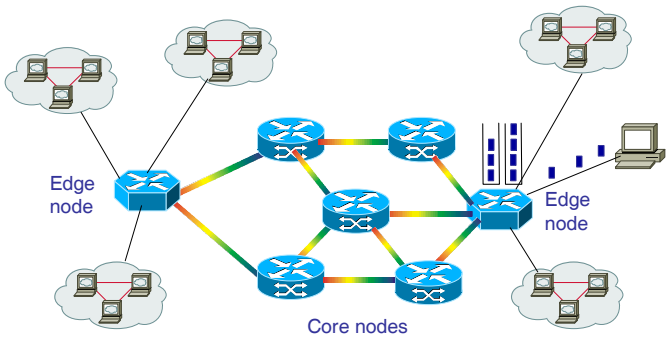


Figure 1: OBS Network architecture.

This work has been carried within the "TCP over OBS" taskforce of the European project NOBEL and the JP on OBS of the Network of Excellence e-Photon/One+, and has been also supported in part by the Spanish Ministry of Education and Science (Ministerio de Educación y Ciencia) under Grant TEC2005-04923.

O. González de Dios is with Telefónica I+D, Emilio Vargas 6, Madrid, (e-mail: ogondio@ tid.es).

I. de Miguel is with the Department of Signal Theory, Communications and Telematics Engineering, University of Valladolid, Campus Miguel Delibes, Camino del Cementerio s/n, Valladolid, 47011, Spain. (e-mail: ignacio.miguel@tel.uva.es).

V. López was with Telefónica I+D when this work was done. He is now with the Networking Research Group (NRG) of Universidad Autónoma de Madrid (e-mail: victor.lopez@uam.es).

One of the issues that have recently received significant attention is the evaluation of the network performance when the characteristics of the upper layers of the protocol tower are considered. TCP [7] is the *de facto* standard in transport protocols, and is used by most of user applications, such as web browsing, e-mail or FTP. Moreover, it is expected that TCP will still hold the leadership in the medium and long term. Hence, is necessary to analyze the performance of TCP over OBS networks, because this combination can be a feasible alternative for the next generation high speed optical Internet. Detti *et al.* first studied the performance of TCP Reno over OBS from an analytical point of view [8]. They assumed a simple network model consisting of only one TCP client and one server connected by a lossy OBS link with a lossless return link. Later, they improved the model by considering a lossy return link [9]. They proved the existence of an effect, which they named *correlation benefit*, which is due to the aggregation process, and that can lead, in some cases, to a significant increase of the TCP transmission rate. In order to check the existence of the correlation benefit in pragmatic scenarios, they also studied a sample scenario with additional traffic sources. While such simulation shows the correlation benefit, quantitative results in that scenario deviate from the analytical model (since that model was obtained for the simple one client-one server scenario). Hence, it becomes clear the necessity of performing both a theoretical and a much more extensive simulation analysis to evaluate the impact of multiple traffic sources on TCP over OBS networks. Yu *et al.* [10-12], compared the impact on performance of different versions of TCP, namely Reno, SACK and New Reno, both analytically and by means of simulation, showing that SACK is the version that usually leads to higher throughput. However, they only considered the simple scenario with only one TCP client and server. Gowda *et al*. [13] presented a number of simulation studies about the impact on throughput and delay of the maximum burst size and timeout when feeding the OBS network with three and 30 TCP sessions, but they provided just a few qualitative explanations of the results. Pleich *et al.* [14] presented some simulation results of TCP Reno over OBS considering up to 900 TCP sources, finding than realistic TCP-controlled traffic on top of OBS is much more robust to burst losses than predicted by other papers. They considered that "*probably the claimed sensibility of throughput to burst losses depends on old (simulated) TCP versions or on too few traffic sources*". On the other hand, Gonzalez *et al.* [15] provided a qualitative explanation, supported by simulation results, showing that the throughput in OBS networks degrades when the delayed ACK feature of TCP is employed. Finally, Guo *et al.* [16] reported some experiments done with TCP over an OBS testbed.

In this paper, we perform an extensive simulation study of the performance of TCP over OBS networks when multiple traffic sources feed a burstifier. Not only quantitative results are presented, but also qualitative explanations of the impact of different scenarios and parameters such as the utilization of different versions of TCP (Reno and SACK), the utilization or not of the delayed ACK algorithm, the impact of the burstification timer, and of course, the comparison of scenarios with a single traffic flow and with additional background traffic.

First, in section II, we review the basic characteristics of TCP, which are required to understand its behavior. Then, in Section III, we explain the most important problems that TCP has in an OBS network, mainly delay and burst losses, supporting our explanations with simulation results. Finally, in Section IV, we analyze the distribution of the TCP segments of a flow when carried by optical bursts, and evaluate the performance of TCP over OBS networks in the scenarios mentioned above.

II. Introduction to TCP

TCP has become the *de facto* standard in transport protocols and it is used by most of user applications [7]. This protocol is in charge of controlling end-to-end communications, using the facilities provided by the network layer, which is usually IP. In this section, we briefly describe the basic behavior of TCP; since, as we will show later, some of the characteristics of TCP explained here have a significant impact on the performance of OBS networks.

A. TCP transmission, flow and congestion control

TCP sends data in chunks, called segments, which are acknowledged by the receiver. Each segment is numbered with the aim of facilitating both reordering in the destination node and detecting lost segments. TCP uses a sliding window mechanism for flow control. Hence, only a certain number of segments are allowed to be sent. When these segments are acknowledged, TCP is allowed to send more data. The *transmission window* determines the maximum number of segments that can be in transit, that is, how many segments can have been sent but have not been acknowledged yet. Each time an acknowledgement is received, the window is updated, and TCP is allowed to send new segments.

Another important concept in TCP is the *round trip time* (RTT), which is defined as the time between the sending of a segment and the arrival of its acknowledgement (ACK). The most common situation is that all the segments of the window are transmitted within a RTT. Thus, an estimate of the TCP transmission rate, $X(t)$, in segments per second, is

$$X(t) = \frac{W(t)}{RTT}, \qquad (1)$$

where $W(t)$ is the size (in segments) of the transmission window at a certain time. Therefore, the maximum TCP throughput is [17]

$$X_{\max} = \frac{W_{\max}}{RTT}, \qquad (2)$$

where $W_{\max}$ is the maximum size of the transmission window.

The transmission window size is determined by the minimum of two limits, one imposed by the receiver, the *reception buffer* or *receiver advertised window*, which indicates the amount of data that it is able to buffer, and another imposed by the sender, the *congestion window*, which is a limit on the segments in transit in order not to overload the network. TCP has several phases in the transmission, where the congestion window varies in a different way. Initially, TCP has a low congestion window, typically one segment, and its size is increased by one segment every time an acknowledgement (ACK) is received. This phase is called *slow start*, and results in an exponential increase of the congestion window, as shown in Fig. 2.a. This behavior ends when the congestion window reaches a certain threshold, *ssthresh* (slow start threshold). The next phase is *congestion avoidance*, where the congestion window is increased at a slower rate, at most one segment per round trip time, hence linearly increasing with time (Fig. 2.a). The purpose of increasing the congestion window at a slower rate is not to overflow the network. Fig. 2.b shows the evolution of the transmission window, which is the minimum of the congestion window and the receiver advertised window (or reception buffer).

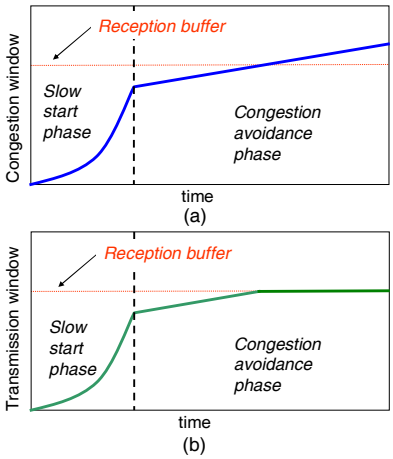


Figure 2: (a) Evolution of the congestion window for a TCP flow. (b) Evolution of the transmission window for a TCP flow.

B. *Reliability of the transmission*

TCP has a number of mechanisms to detect and recover from segment losses, thereby providing transmission reliability. TCP was created with the idea that the loss of a segment is a clear indication of congestion (something which is not usually true in an optical network). Thus, when a segment loss is detected, the aim of the recovery mechanisms is not only to retransmit the lost segment, but also to slow down the transmission rate (by means of reducing the congestion window) in order to avoid further network congestion.

TCP has two ways of detecting losses, by the triggering of a retransmission timeout and by means of the reception of three duplicate ACKs. As we previously said, all segments must be acknowledged. When a segment is sent, a timer called *retransmission timer* is set. If the timer expires before the arrival of the ACK, it is assumed that the segment was lost. Thus, when this timer expires, the lost segment is retransmitted, and the transmission goes to the beginning of the slow start phase. Thus, the TCP transmission rate is drastically reduced. The second method to detect losses is the reception of duplicate ACKs. When a destination TCP node receives a segment whose sequence number is not the expected one, but a higher one, so that one segment may have been lost, it must send a duplicated ACK immediately, confirming again the segments already acknowledged. When three duplicated ACKs (four identical ACKs) are received by the TCP source, the segment is retransmitted without waiting for the timer expiration. This mechanism is called *fast retransmit* [18]. Sometimes, depending on the TCP version employed, this mechanism is combined with *fast recovery* [18]. When fast recovery is used, the TCP sender reduces the congestion window to roughly half of the size of the transmission window (additional details can be found in [18]). In this way, the reduction of the transmission rate is not as drastic as in the case of the timer expiration. Fast retransmit and fast recovery are efficient methods when only one segment is lost, but they are not so efficient when several consecutive segments are lost. This fact will be shown and explained in Section III, where we analyze its impact in OBS networks.

With the aim of improving the performance in the case of multiple segment losses, a technique called *selective acknowledgement* (SACK) has been proposed [19, 20]. The selective acknowledgment is an ACK which includes an additional field where, if the segments arrive out of order, the receiver tells which segments have arrived out of order and how many consecutive segments have been received. In this way, the sender is able to know how many segments have been lost and how many have arrived correctly. Hence, the sender can retransmit all the lost segments without waiting for the timer expiration.

There are a number of TCP versions such as Tahoe, Reno, New Reno, SACK, and Vegas. The main differences among them are the algorithms that they employ when congestion is detected. In this paper we focus on the most important TCP versions nowadays: TCP Reno, which implements fast retransmit and fast recovery, and TCP SACK, which also uses those algorithms together with selective acknowledgement.

C. *Delayed ACK*

The first version of the TCP protocol [21], states that a TCP receiver must send an acknowledgement for each incoming segment. This behavior was later modified by RFC 1122 [22], which specifies the *delayed ACK* algorithm. In the event of an incoming segment, the TCP receiver does not immediately send an acknowledgement. Specifically, the ACK is delayed until a second segment arrives or a timer expires. Delayed ACK was introduced to reduce the load in the network. However, this improvement is only significant in asymmetric networks. Moreover, a TCP sender increases its congestion

window according to the number of ACKs received, not to the bytes acknowledged. Hence, when delayed ACK is not used, the congestion window increases faster, leading to higher TCP transmission rates [23].

III. Impact of the Burstification in TCP

Since in an OBS network, packets are aggregated into bursts before being transmitted, the performance of TCP differs from that in ordinary packet networks. First of all, packets suffer an additional delay in the transmitter due to the burstification process. Secondly, when a burst is lost, several segments belonging to the same TCP connection may be lost. Hence, OBS networks are more prone to suffer the lost of *consecutive* segments than packet networks.

A. Impact of delay penalty

IP datagrams, which contain TCP segments, are grouped into bursts in the ingress nodes of the OBS network. The assembly of the datagrams into bursts introduces an additional delay, due to the waiting time until the burst is completed. Hence, the RTT is increased, and therefore the TCP throughput (see equation 1) is reduced. While being assembled, a TCP segment has to wait between 0 and T_b until the burst is completed (being T_b the burst aggregation time). As the segment containing data waits at maximum T_b, and its ACK also waits at maximum T_b before being sent back, in the worst case, the RTT will be increased by $2T_b$ [11].

B. Impact of burst losses

The most influential aspect on TCP performance is the loss of bursts due to contention at intermediate nodes. The loss of a burst generally implies the loss of several consecutive TCP segments belonging to the same flow. As mentioned in Section II, TCP has two ways to detect segment losses; either by means of the reception of duplicated ACKs, or by means of the expiration of a retransmission timeout. In OBS, bursts contain several segments; hence, if a burst is lost, depending on the number of segments it contains (when compared to the size of the transmission window), TCP will have a different reaction. If a burst which contains a complete transmission window is lost, the retransmission timer will expire and the transmission will switch to the slow start phase. On the other hand, if the lost burst does not contain a complete window, out of order segments will arrive at the receiver, so duplicated ACKs will be sent, and the mechanisms of fast retransmit and fast recovery will enter into action.

It is important to remark that in a pragmatic OBS network, a burst will carry TCP segments associated to different TCP connections or flows. This is due to the fact that different sources will be feeding the burstifiers of the edge nodes. Therefore, in case of the loss of a burst, the influence on the performance of TCP depends on the total number of segments *belonging to the same flow* that are carried by that burst. In order to analyze the impact of this issue from a qualitative point of view, we have performed a number of simulation experiments in OPNET Modeler for TCP Reno and SACK. The main results can be summarized in three scenarios[1].

1) Loss of a burst carrying only one segment of a TCP flow

Let us assume that a burst is transmitted from an edge node A to another B. The burst only carries one segment from a TCP flow, and unfortunately, the burst is lost.

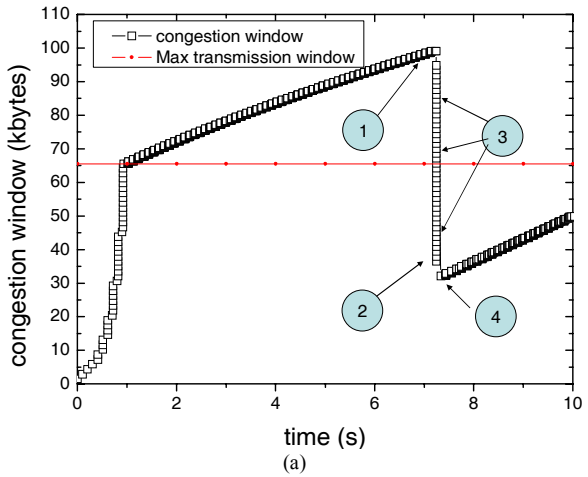
Fig. 3 shows the evolution of the congestion window for TCP Reno for this scenario. Fig. 3.a shows the complete transmission, while Fig. 3.b shows a zoom around the instant when the burst loss takes place. In those figures, we have numbered a few noteworthy points of the simulation. At point ①, the loss of the burst occurs. That burst only contains one segment of the TCP flow that we are analyzing. If the transmission window was big enough, additional segments will have been sent in other bursts after that lost one. Hence, if at least three segments reach the destination node after the loss, the TCP sink (at node B) will detect that they are out of order, so it will send three duplicated ACKs (one for each out of order segment). When the third duplicated ACK reaches node A, the TCP source will trigger the fast retransmit and fast recovery mechanisms. First of all, the congestion window is reduced to *fligthsize*/2 + 3 (point②). *Flightsize* is the number of segments that have been sent but have not been acknowledged yet, that is, the number of segments in transit, and its value usually matches with that of the transmission window. Then, the segment of the flow which was lost (due to the loss of the burst) is retransmitted, and the congestion window is increased every time a new duplicated ACK arrives (points ③). Finally, when the ACK that acknowledges the retransmitted segment arrives, the congestion window is reduced to the half of *flightsize* (point④).

In this scenario, the duplicated ACKs arrive at the TCP source before the expiration of the retransmission timer. Thus, the reduction of the transmission rate is not as significant as if the timer had expired.

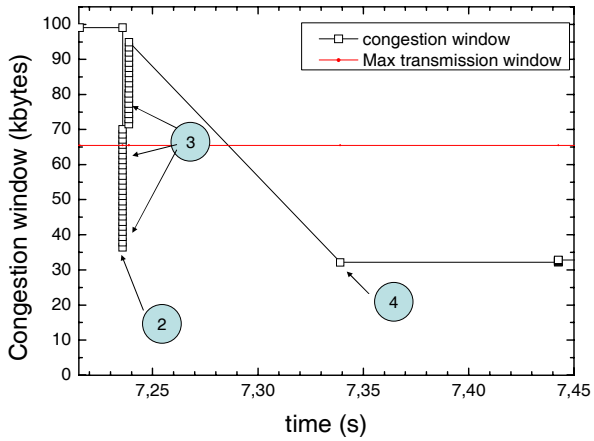
The behavior of TCP SACK is shown in Fig. 4. After the burst loss (point ①), when the three duplicated ACKs arrive, the congestion window is reduced to the half of *flightsize* (point②), and the lost segment is retransmitted. Then, new duplicated ACKs arrive (points ③). One RTT after retransmitting the segment, its ACK arrives confirming the reception at destination, and then the size of the congestion window keeps growing according to the congestion avoidance phase (point④).

Therefore, both TCP Reno and SACK recover from the segment loss in a short time, one RTT after receiving the three duplicated ACKs. After the recovery, the congestion window is reduced to the half of *flightsize*. Hence, in this scenario, the behavior of both versions is very similar.

[1] There are additional scenarios (or subscenarios) to those described here, but these are the most relevant ones.



(a)



(b)

Figure 3: Evolution of the congestion window for TCP Reno when a burst containing one segment of the flow is lost. (a) Complete transmission. (b) Zoom around the burst loss.

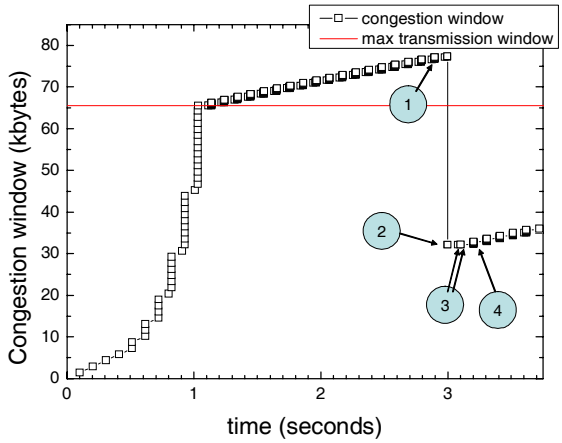


Figure 4: Evolution of the congestion window for TCP SACK when a burst containing one segment of the flow is lost.

2) Loss of a burst carrying two or more segments of a TCP flow

Now, we study the case where a burst carrying two segments of a TCP flow is lost, and the following burst (with new segments) arrives correctly. Figs. 5 and 6 show the behavior of TCP Reno and TCP SACK, respectively.

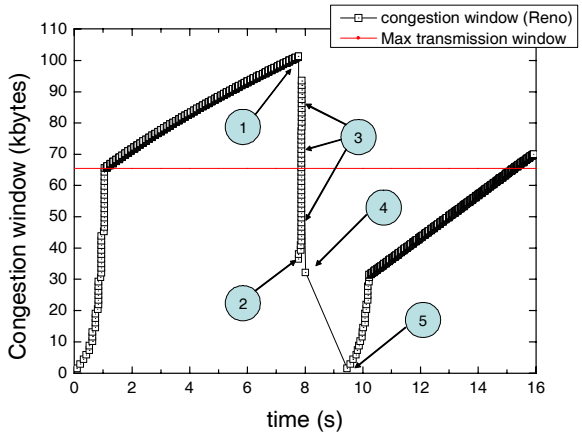


Figure 5: Evolution of the congestion window for TCP Reno when a burst containing two segments of a flow is lost.

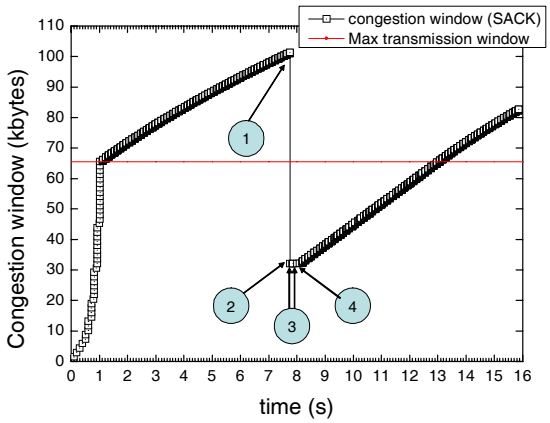


Figure 6: Evolution of the congestion window for TCP SACK when a burst containing two segments of a flow is lost.

In the case of TCP Reno (Fig. 5), after a burst loss detection by means of duplicated ACKs (point ①), the sender retransmits the first segment lost and reduces the congestion window to *fligthsize*/2 + 3 segments (point ②). Then, as new duplicated ACKs arrive, the congestion window is increased by one segment (points ③). In the example shown in the figure, the sender has in transit as many segments as the receiver advertised window. Hence, no matter the value of the congestion window, the TCP sender cannot send any new data. If the receiver advertised window had been higher than the congestion window when fast recovery started, the sender could have been allowed to send new segments in some of these window updates (but this is not the case considered in the example). Then, the ACK of the first lost segment (which was generated after the reception of the retransmitted segment) arrives, and the congestion window is reduced (point④). At this point, in the example, the number of segments in transit is reduced just by one (thanks to the segment that has arrived correctly, but not more, because the second segment is lost and the rest of the segments are still unacknowledged). Hence, the congestion window does not allow to send new segments, so the receiver cannot send three duplicated ACKs. Therefore, the sender remains inactive until the retransmission timer

associated to second lost segment expires (point⑤). Then, the segment is retransmitted and the transmission phase is slow start again. In general, TCP Reno usually has to wait for the retransmission timeout to recover from a two-segment loss, and almost always in case of a three or more segments loss. To be precise [20] states that "*when two packets are dropped from a window of data, the Reno sender is forced to wait for a retransmit timeout whenever the congestion window is less than 10 packets when Fast Recovery is initiated, and whenever the congestion window is within two packets of the receiver's advertised window when Fast Recovery is initiated*".

In the case of SACK, after a burst loss detection by means of duplicated ACKs (point①), the congestion window is reduced to *flightsize*/2 (point ②). As new duplicated ACKs arrive, the congestion window remains constant. These acknowledgements contain selective ACK information, so that using that information, all the consecutive lost segments can be retransmitted. When the ACKs confirming the reception of these segments arrive, the transmission continues with the congestion avoidance phase (point④).

The case we have explained corresponds to the loss of a burst with two segments belonging to a same TCP flow, but if the burst has more than two segments, the behavior is similar (whenever three duplicated ACKs arrive).

In this scenario, TCP Reno almost always recovers with a timer expiration and continues transmission with slow start, thereby reducing drastically the transmission rate. Nevertheless, SACK recovers in approximately one RTT and continues the transmission with just the window halved instead of reducing it to one segment, like in the Reno case. Therefore, SACK offers an important improvement of performance in this scenario. This is the expected scenario in OBS networks, hence the use of SACK is highly advisable in these networks.

3) Loss of a burst carrying a complete window of a TCP flow

If a burst contains all the segments sent in a TCP transmission window, its loss implies the expiration of the timer. This is because the TCP sender (in both Reno and SACK), does not receive any duplicated ACK (Figure 7).

Therefore, in summary, depending on the number of segments lost in each burst, and the TCP version, TCP will recover in a different way. TCP SACK presents benefits over Reno in case multiple segment losses, except when the full transmission window is lost.

C. Classification of traffic sources

As shown in the previous analysis, the number of segments belonging to the same flow that are transported by each burst has an impact on the performance of TCP over OBS networks. In previous works, three different types of traffic sources have been defined [8, 11-12]: fast, medium and slow flows. The classification of a flow in one of the three categories depends on the access rate (*BW*, measured in segments per second), on the assembly period (T_b, in seconds), and on the maximum transmission window ($W_{\max}$, in segments):

- A fast flow verifies that $BW \cdot T_b \geq W_{\max}$. This means that the whole transmission window is transmitted in a single burst.
- A medium flow fulfills $1 < BW \cdot T_b < W_{\max}$. Hence, the content of a TCP transmission window is sent in several bursts.
- A slow flow verifies that $BW \cdot T_b \leq 1$. Thus, each burst contains only one segment.

For the medium flow case, the maximum number of segments per burst is $S_{\max} = BW \cdot T_b$. Hence, the number of bursts that will carry $S_{\max}$ segments is $N_{\max} = \lfloor W_{\max} / S_{\max} \rfloor$, and the last burst will contain $S_{last} = \mathrm{mod}(W_{\max} / S_{\max})$ segments. In total, there will be $N_{total} = \lceil W_{\max} / S_{\max} \rceil$ bursts. Therefore, the mean number of segments per burst is:

$$S = \begin{cases} W_{\max}, & \text{for fast flows} \\ \dfrac{N_{\max} \cdot S_{\max} + S_{last}}{N_{total}}, & \text{for medium flows.} \\ 1, & \text{for slow flows} \end{cases} \quad (3)$$

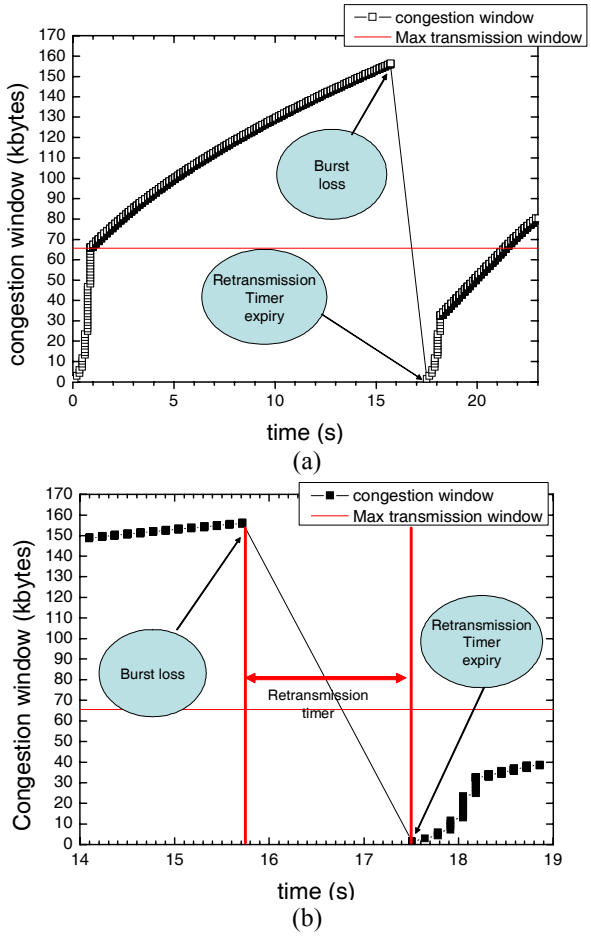


Figure 7: Evolution of the congestion window when a burst containing a complete transmission window of the flow is lost. (a) Complete transmission (b) Zoom around the burst loss.

However, it is important to remark that this classification is based on a model with only one TCP client and server, so care must be taken to avoid pitfalls. In the next section, we will show, by means of simulations, that when additional traffic is considered, the number of segments of a single TCP flow being carried by a burst is reduced.

D. *Amplification effect*

One of the benefits of OBS is the so called *amplification effect* or *correlation benefit* [9-10]. Let us assume that the burst loss probability is 10^{-2} and that we want to transmit 1000 segments. If every burst carried five segments, we would need 200 bursts. Thus, in mean, two bursts would be lost. If, for instance, TCP Reno is used, this would mean that after each loss event, the transmission would go to the slow start phase. On the other hand, if every burst carried 10 segments, then 100 bursts would be needed. In this case, only one burst (in mean) would be lost. The behavior of TCP is basically the same when either five or ten consecutive segments are lost. Thus, TCP Reno would also switch to the slow start phase. However, as only one burst is lost in this second situation, it would only run once into that phase. Hence, this situation would lead to higher throughput than the previous one. So, as the number of segments per burst increases, the throughput increases as well. This is the correlation benefit.

IV. Simulation Study of TCP over OBS with and without Multiple Data Flows.

With the aim of evaluating the performance of TCP over OBS networks from a quantitative point of view, a simulation model has been developed using OPNET Modeler 11.0 [24].

A. *Simulation models*

We have developed two simulation models. The first model (Fig. 8) is based on the classic models used in the literature to study TCP over OBS [8], [11-12]. Such a model only has one TCP client and one server. Both transmission ways are modeled, so that the ACKs are also assembled into bursts and transmitted through a lossy OBS link. The second model (Fig. 9) is more pragmatic. The OBS burstifier is not only fed by the TCP client (or server), but also by additional traffic, which is created by means of fractal traffic generators [25]. The fractal traffic generators provide an average rate of 900 packets per second. The length of those packets is exponentially distributed with average of 1024 bits, and the Hurst parameter is 0.7. The rest of the simulation parameters employed in the model have been set as shown in Figs. 8 and 9. Moreover, in Table I, we show the most important TCP parameters which have been used in the simulation.

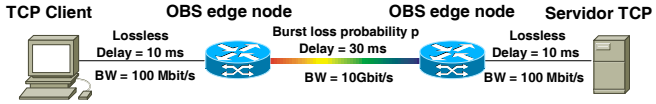


Figure 8: Simulation model of TCP over OBS with only one TCP flow.

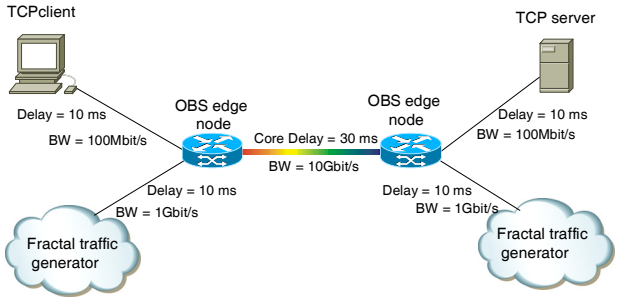


Figure 9: Simulation model for TCP over OBS with one TCP flow and additional traffic sources.

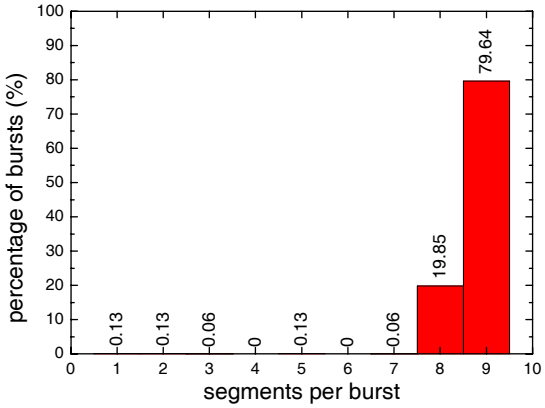
Parameter	Value
Maximum segment size	1460 bytes
Maximum transmission window	65535 bytes (44 segments approx.)
TCP Version	Reno and SACK
Delayed ACK	When used, timer set to 200 ms
Minimum retransmission timer	1 second

TABLE I: TCP PARAMETERS

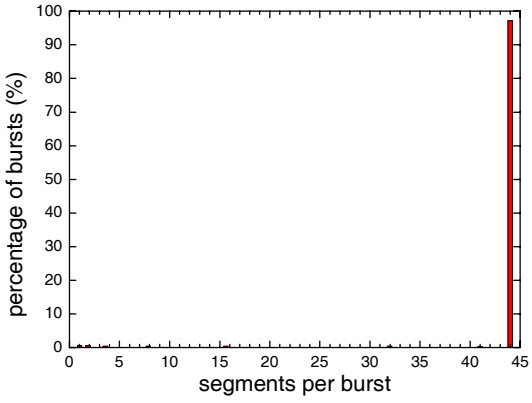
B. *Study of the number of TCP segments per burst*

First of all, we have studied the number of segments of a TCP flow carried by a burst (S), since it is a critical issue as shown in Section III. For that aim, we have simulated the transfer of files of 20 Mbytes through a lossless OBS network; first, without any additional traffic, and then, together with additional traffic. 500 file transfers have been simulated, and the histogram of the number of segments of a TCP flow per burst has been calculated. The burst aggregation timer was set to two different values, 10 ms (which corresponds to a fast flow scenario) and 1 ms (which corresponds to a medium flow scenario).

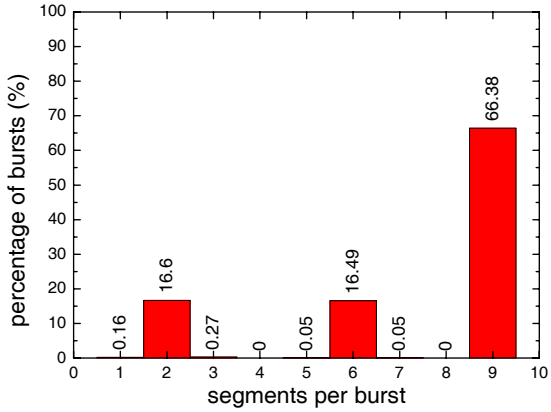
Firstly, we have simulated the simple scenario (Fig. 8) without delayed ACK. Figs. 10.a and 10.b show the results for a medium and a fast source, respectively. The result, as expected, is very deterministic, and shows that most of the bursts carry 8 or 9 segments for the medium source, and 44 segments for the fast source. For instance, in the latter case, all the segments of the transmission window (44 as stated in Table I) are aggregated in a burst and transmitted. Then, the ACKs for all those segments are aggregated in another burst and sent back to the TCP client, and so on. However, when the delayed ACK algorithm is switched on, the results (Figs. 10.c and 10.d) deviate from that behavior. When delayed ACK is used, the TCP receiver does not immediately sends an acknowledgement when a segment is received, but the ACK is delayed until a second segment arrives or a timer expires. That behavior leads to a certain amount of "fragmentation" of the sequence of data segments and acknowledgements (they are not always sent in a row, as it happened in the previous simulations). For this reason, the histogram presents peaks for other values of S, mainly at two and multiples of two, due to that feature of acknowledging the second segment.



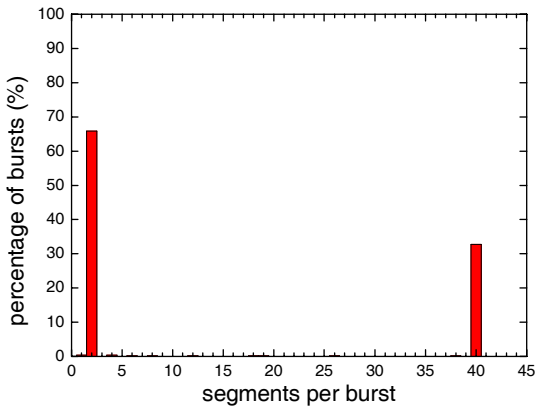
(a) **one flow** and **without delayed ACK** for a **medium** flow.



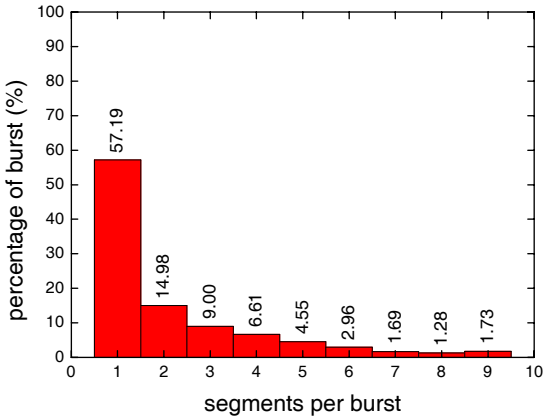
(b) **one flow** and **without delayed ACK** for a **fast** flow.



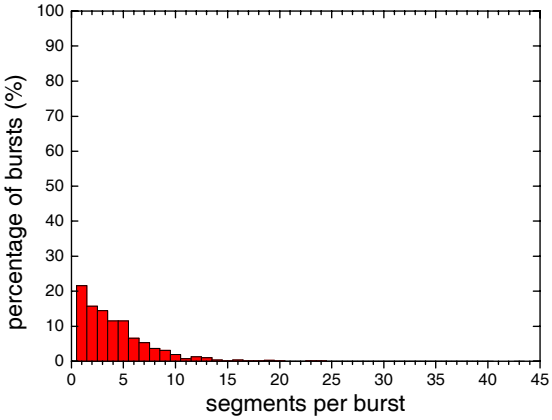
(c) **one flow**, **with delayed ACK** for a medium flow



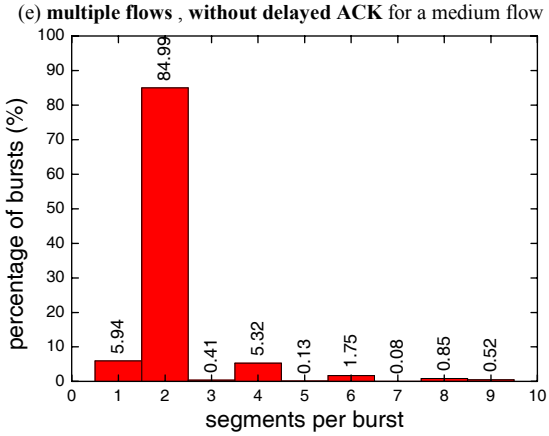
(d) **one flow** and **using delayed ACK** for a **fast** flow



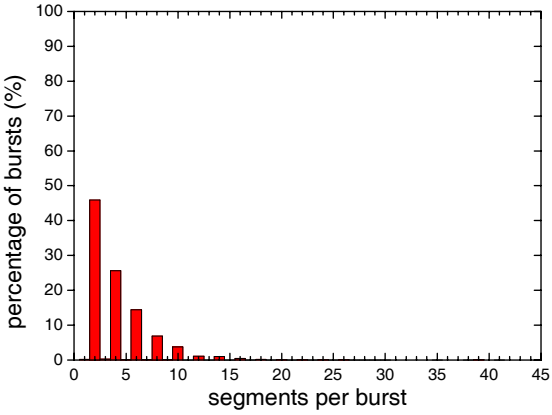
(e) **multiple flows** , **without delayed ACK** for a medium flow



(f) **multiple flows** , **without delayed ACK** for a medium flow



(g) **multiple flows** and **using delayed ACK** for a medium flow



(h) **multiple flows** , **without delayed ACK** for a medium flow

Figure 10: Histograms of the number of segments of a TCP flow per burst for different scenarios.

When background traffic is added, the results are completely different (Figs. 10.e, 10.f, 10.g, 10.h). The bursts generally carry a lower number of TCP segments of a flow than in the previous scenarios. In this case, as the formation of a burst can be initiated not only by the flow which is being studied, but by any of the packets of the other traffic sources, there is a significant fragmentation of the sequence of data segments, as they will be sent in different bursts. Like in the case with no additional traffic, if we switch the delayed ACK on, the number of segments per burst is mainly distributed over multiples of two.

To explain in more detail the impact of adding background traffic, we will refer to Figs. 11 and 12. The former shows the sequence number of the segments sent by the TCP sender *without* background traffic for a fast flow without delayed ACK. It is clear that the transmission is very bursty and synchronized. Every RTT, all the segments of the TCP window are transmitted in a row, so all these segments are assembled into a single burst. Fig. 12 shows the case where the burstifier is fed with additional traffic. The transmission begins with the same behavior: all segments of the TCP window are assembled together in a burst. However, as time goes by, the transmission of all consecutive segments in a row is broken. For instance, a packet of another flow arrived at the burstifier and started the process of burst formation, so that when the row of segments of the TCP flow that we are considering arrived at the burstifier, a portion of them entered in the buffer before the expiration of the burst aggregation timer, while the rest arrived after the expiration, and then should be transmitted in a different burst. Hence, the transmission of the segments is finally spread along a RTT, and each burst contains a lower number of segments of the flow. Therefore, after several transmission rounds, the mean number of segments per burst (S) reaches a steady state (Fig. 13 shows the moving average). In these simulations we used a timer-based assembly algorithm. If an algorithm based on burst size or a hybrid one were used, we expect the results to be even more noticeable.

In order to be able to determine the throughput of TCP in OBS networks, and to find what the real correlation benefit is, it is necessary to know the average number of segments per burst. We have measured this value for four scenarios, and for different burst aggregation timers. Fig. 14 shows the simulation results as well as the analytical results obtained for a single TCP flow without additional traffic (equation 3). The simulation shows that when additional traffic is considered, the mean number of TCP segments of a TCP flow per burst is much lower than in the case of the single flow. Regarding the utilization or not of the delayed ACK feature, the differences are not significant when additional traffic is considered, but they are in the simple scenario. In summary, the results show that the number of segments per bursts is highly overestimated if we try to apply the results obtained when the burstifier is fed by only one TCP flow, to a pragmatic scenario with multiple traffic sources.

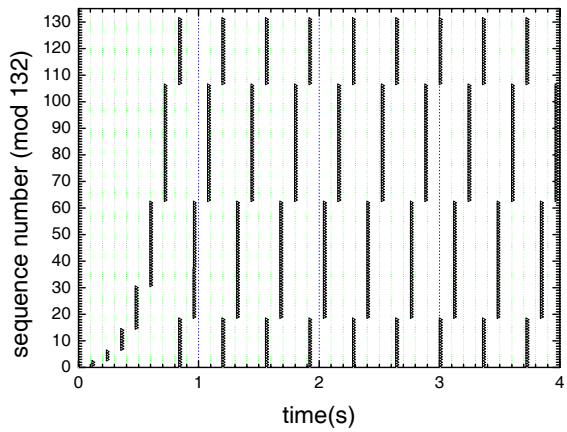


Figure 11: Packet trace, transfer of a single flow

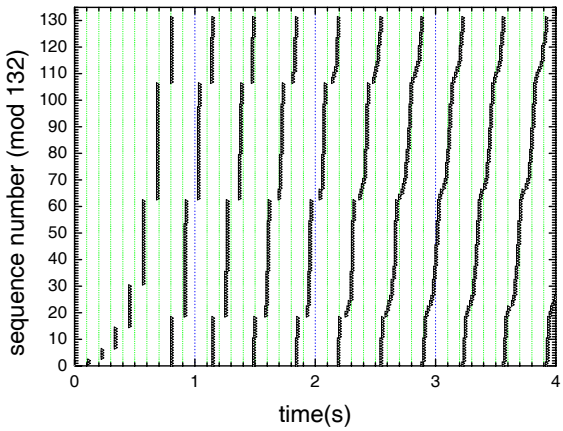


Figure 12: Packet trace, transfer of a TCP flow in the model with additional traffic

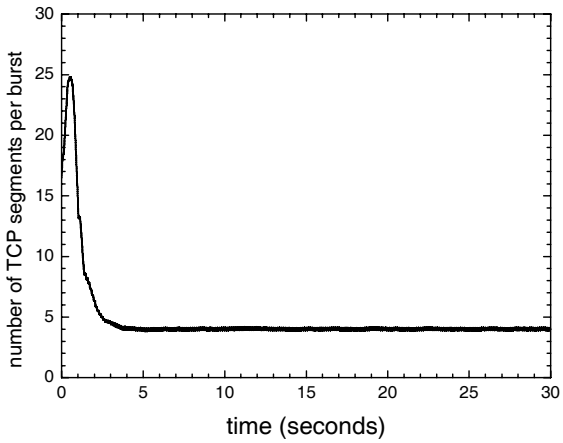


Figure 13: Moving average of the number of segments per burst of a TCP flow in a transmission.

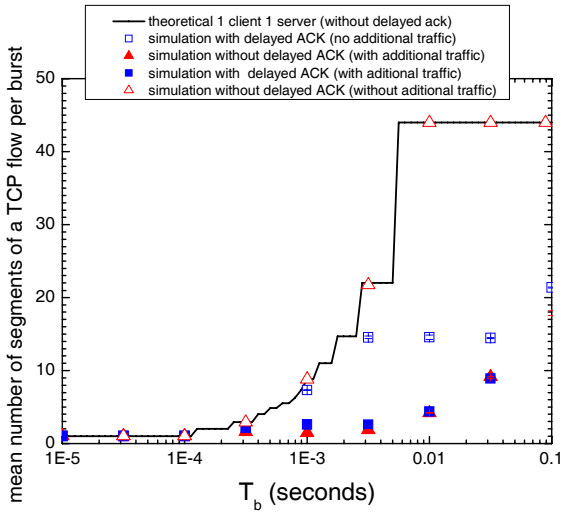


Figure 14: Average number of segments of a TCP flow per burst.

C. *Simulations for different assembly timers*

Now, our aim is to quantify the impact on goodput, which is defined as the useful data received at destination per unit of time. The simulations have consisted in the transfer of one hundred files of 20 Mbytes, setting the burst loss probability to 10^{-3}. Higher probabilities would not be realistic, since an operator would not deploy a network with such high losses. The values of the timers have been chosen in the range of previous studies. The simulations results are shown in Figs. 15 and 16. The average values are represented together with the 95% confidence interval. A number of conclusions can be obtained from these figures.

TCP Reno vs. TCP SACK: The performance of TCP Reno and SACK for low burstification timers (10^{-5}-10^{-4}), and for high timers (around 100 ms) is very similar. For low timers, the number of segments per burst is one, hence all losses are equally solved by TCP Reno and SACK, as we explained in Section III. For high timers, the lost of a burst usually means the lost of the complete window of the TCP flow, and, again, as described in Section III, both versions of TCP have the same behavior. However, for the mid timers, SACK offers better performance due to its capacity to recover from multiple segment losses. Therefore, the results completely agree with the qualitative analysis of Section III.

Results with and without delayed ACK: As we demonstrated in [15], the goodput when the delayed ACK algorithm is employed, is lower than when that algorithm is not activated. The qualitative explanation was given in Section II.C, and can be found in more detail in that reference.

Results with and without additional traffic: The results show that for low timers there are no differences in performance when considering or not additional traffic. However, for timers from 0.3 to 10 ms, the goodput is higher for the scenario with no additional traffic than for the scenario with background traffic. This is due to the fact that the number of segments per burst belonging to the same flow is higher when additional traffic is not considered, thereby leading to a higher correlation gain. On the other hand, for high timers, the bursts carry a very high number of segments when no additional traffic is considered, so the losses usually lead to a timeout event. Hence, the losses are more harmful when no additional traffic is considered for these high timers.

Optimal value of the burstification timer: The goodput depends on the value of the burstification timer. The higher goodput is obtained for timers from 0.3 to 10 ms. This is due to the correlation gain. However, if the timer is set to a higher value, the goodput decreases, being even lower than that obtained for very low timers where there is not correlation benefit. The reason for that behavior is that not only burst losses affect the performance of TCP over OBS, but also the delay penalty imposed by the burstifier (Section III.A).

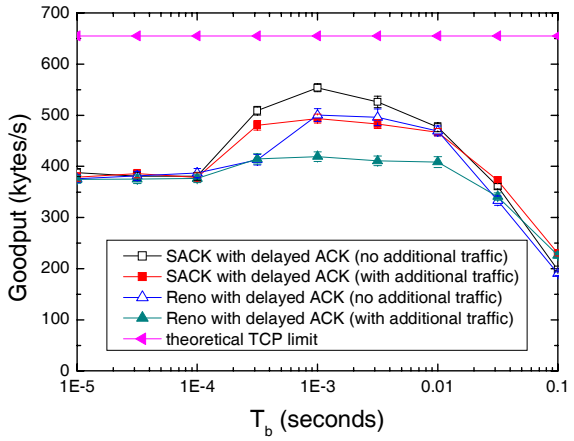


Figure 15: TCP goodput for TCP Reno and SACK, with and without additional traffic, with p =10^{-3}, when the delayed ACK algorithm is used.

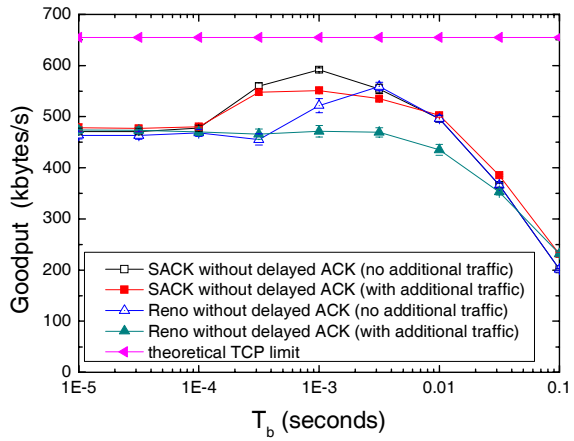


Figure 16: TCP goodput for TCP Reno and SACK, with and without additional traffic, with p =10^{-3}, when the delayed ACK algorithm is not used.

In order to evaluate the impact of the delay penalty introduced by the burstifier, we define the following figure of merit:

$$F_{delay} = \frac{\text{Goodput}\,(T_b)\ \text{without losses}}{\text{Goodput}\,(T_{b\_ref})\ \text{without losses}}, \qquad (4)$$

where T_{b_ref} is a value of the burstifier timer so small that the bursts always contain one segment, and the delay in the burstifier is very small when compared to the *RTT*. We have chosen 10^{-5} seconds as it corresponds to the lowest value of the timer used in the simulations.

As the TCP Send Rate is inversely proportional to *RTT*, and the maximum increase in the *RTT* is $2T_b$, it is expected that the figure of merit will be, approximately,

$$F_{delay} = \frac{RTT_0}{RTT_0 + 2T_b}. \qquad (5)$$

where RTT_0 is the basic round trip time of the network, that is, the round trip time when no burstification delay is considered.

Fig. 17 shows both the analytical figure of merit (equation 5) and the simulation results. They correspond to the networks shown in Figs. 8 and 9 (RTT_0 = 100 ms), when different scenarios regarding to delayed ACK and background traffic are considered. In this case, it is important to remark that since there are no losses, there is no difference between TCP Reno and SACK. As in the previous simulations, we have measured the transfers of files of 20 Mbytes to obtain the experimental results. The simulation results match with the expected figure of merit of the delay penalty. In this case, the difference between considering additional traffic or not, is almost negligible. The most significant conclusion is that values of the burstification timer lower than 10 milliseconds do not harm TCP goodput. However, for higher values, there is significant performance degradation. Hence, high burstification timers are very influenced by delay penalty, and that is why the goodput significantly decreases for those timers in Figs. 15 and 16.

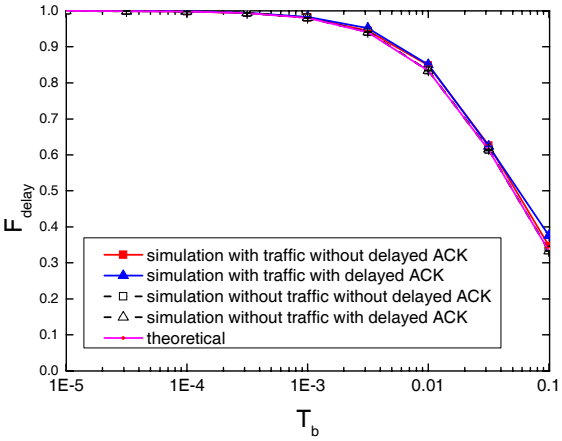


Figure 17: F_{delay}: Figure of merit of the delay penaly for RTT_0=100ms

V. SUMMARY

We have performed a simulation study of the performance of TCP over OBS networks when multiple traffic sources feed a burstifier. The simulations have been used with a double purpose, to present quantitative results, but also to provide qualitative explanations of the processes and factors with an impact on performance.

It has been shown that when additional traffic is considered, the correlation gain is not as high as when only one TCP flow is sent through the network. The reason is that the bursts carry a lower number of segments belonging to the considered TCP flow. Moreover, it has been confirmed that TCP SACK is highly beneficial for OBS networks, as it leads to higher goodput than TCP Reno, and that if the delayed ACK feature is activated, it leads to lower goodput. This latter fact had been demonstrated for the single flow case, but not for the case where multiple traffic sources share a burstifier.

We are currently working in an analytical model to complement and enhance the simulation results. Also, in future work, the performance will be analyzed considering the utilization of larger transmission windows, as well as new TCP versions.

REFERENCES

[1] R. Ramaswami and K. N. Sivarajan, *Optical Networks: A Practical Perspective*, Morgan Kauffman Publishers, Inc., 1998.

[2] T. S. El-Bawab and J.-D. Shin, "Optical Packet Switching in Core Networks: Between Vision and Reality", *IEEE Communications Magazine*, vol. 40, no. 9, pp. 60-65, Sep. 2002.

[3] Y. Chen, C. Qiao and X. Yu, "Optical Burst Switching: A New Area in Optical Networking Research", *IEEE Network*, vol. 18, no. 3, pp. 16-23 May/June 2004.

[4] Y. Xiong, M. Vandenhoute and H. Cankaya, "Control Architecture in Optical Burst-Switched WDM Networks", *IEEE Journal on Selected Areas in Communications*, vol. 18, no. 10, pp. 1838-1851 Oct. 2000.

[5] X. Yu, Y. Chen and C. Qiao, "Study of Traffic Statistics of Assembled Burst Traffic in Optical Burst Switched Networks", *Proc. Opticomm 2002*, pp. 149-159.

[6] M. Düser and P. Bayvel, "Burst aggregation control and scalability of wavelength-routed optical-burst-switched (WR-OBS) networks", *Proc. ECOC 2002*, vol. 1, pp. 1-2.

[7] R. Stevens, *TCP/IP Illustrated, Volume 1: The Protocols*, Addison-Wesley, 1994.

[8] A. Detti and M. Listanti, "Impact of Segments Aggregation on TCP Reno Flows in Optical Burst Switching Networks", *Proc. of INFOCOM 2002*, vol. 3, pp. 1803-1812.

[9] A. Detti and M. Listanti "Amplification effects of the send rate of TCP connection through an optical burst switching network", *Optical Switching and Networking*, vol. 2, no. 1, pp 49–69, 2005

[10] X. Yu, Y. Chen and C. Qiao, "Study of Traffic Statistics of Assembled Burst Traffic in Optical Burst Switched Networks", *Proc. Opticomm 2002*, pp. 149-159.

[11] X. Yu, C. Qiao, Y. Liu and D. Towsley, "Performance Evaluation of TCP Implementations in OBS Networks", *Technical Report 2003-13*, CSE Dept., SUNY, Buffalo, 2003

[12] X. Yu, C. Qiao and Y. Liu, "TCP Implementations and False Time Out Detection in OBS Networks", *Proc. of INFOCOM 2004*, pp. 774-784.

[13] S. Gowda, R. K. Shenai, K. M. Sivalingam, and H. C. Cankaya, "Performance Evaluation of TCP over Optical Burst-Switched (OBS) WDM Networks", *Proc. IEEE ICC 2003*, pp. 1433–1437.

[14] R. Pleich, M. de Vega Rodrigo and J. Goetz, "Performance of TCP over Optical Burst Switching Networks", *Proc. ECOC 2005*, vol. 4, pp. 883-884.

[15] O. González, I. de Miguel, N. Merayo, P. Fernández, R. M. Lorenzo and E. J. Abril, "The Impact of Delayed ACK in TCP Flows in OBS Networks", *Proc. of NOC 2005*, pp. 367-374.
[16] H. Guo, J. Wiu, J. Lin and Y. Li, "Multi-QoS Traffic Transmission Experiments on OBS network testbed", *Proc. of ECOC 2005*, vol. 3, pp. 601-602.
[17] M. Hassan and R. Jain, *High Performance TCP/IP Networking: Concepts, Issues, and Solutions*, Prentice-Hall, 2003.
[18] M. Allman, V. Paxson and W. Stevens, "TCP Congestion Control", *RFC 2581*, April 1999.
[19] M. Mathis, J. Mahdavi, S. Floyd and A. Romanow, "TCP Selective Acknowledgment Options", *RFC 2018*, October 1996.
[20] K. Fall and S. Floyd "Simulation-based Comparisons of Tahoe, Reno and SACK TCP", *ACM SIGCOMM Computer Communication Review*, vol. 26, pp. 5-21, July 1996.
[21] J. Postel, "Transmision Control Protocol", *RFC 793*, IETF, Sep. 1981
[22] R. Braden, "Requirements for Internet Hosts – Communication Layers", *RFC 1122*, Oct. 1989.
[23] M. Allman, "On the Generation and Use of TCP Acknowledgments", *ACM SIGCOMM Computer Communication Review*, vol. 28, pp. 4-21, Oct. 1998.
[24] OPNET Modeler. http://www.opnet.com
[25] B. Ryu and S. Lowen, "Fractal Traffic Models for Internet Simulation", *IEEE Int'l Symposium on Computer Communications*, pp. 200-206, July 2000.