The impact of delayed ACK in TCP flows in OBS networks

Óscar González1,2, Ignacio de Miguel2, Noemí Merayo2, Patricia Fernández2, Rubén M. Lorenzo2, Evaristo J. Abril2

1 Innovation Strategy, Telefonica I+D, Emilio Vargas 6, 28043 Madrid, Spain
Tel: +34 91 3374013, Fax: +34 91 5103308, E-mail: ogondio@tid.es
2 Dpt. of Signal Theory, Communications and Telematic Engineering, University of Valladolid, Campus Miguel Delibes, 47011 Valladolid, Spain
Tel: +34 983423665, Fax: +34 983423667, E-mail: ignacio.miguel@tel.uva.es

Optical Burst Switching (OBS) has been receiving increasing attention by the research community. OBS networks have unique characteristics that affect the behaviour of the transport layer. Since it is expected that TCP will remain as the prevailing transport protocol, the performance of TCP over OBS networks needs to be studied. There are several papers which have studied, both analytically and by means of simulation, TCP over OBS performance. However, these studies have not considered the delayed ACK feature. In this paper, we analyse how delayed ACK affects TCP performance in OBS networks. We give a qualitative explanation of why TCP throughput is degraded in OBS networks when delayed ACK is used, and support our comments with simulation results.

1. Introduction

Optical Burst Switching (OBS) is a novel technology for the next generation optical Internet that lies between circuit switching (OCS) and packet switching (OPS). It provides a finer granularity than OCS, and its practical implementation with state-of-the-art optical technology is feasible, unlike OPS, which has proven to be currently unaffordable.

In OBS, incoming packets are assembled into bursts in the edge of the network. Prior to the transmission of a burst, a Burst Control Packet (BCP) is sent, which configures all the intermediate nodes. When the optical burst is transmitted, it travels through the core of the network without any conversion to the electrical domain at intermediate nodes. When it finally arrives at the destination edge node, it is disassembled into packets [1].

It is expected that TCP will remain as the main transport protocol in the short and medium term. Thus, the study of TCP performance in OBS networks is a key issue. The main goals are to find the TCP version and TCP parameters that achieve the best performance, and to determine whether current TCP implementations are suitable for such networks or if novel TCP extensions or even a completely new transport protocol are needed. In this paper, the contribution to such goals is to determine whether delayed ACK is a suitable algorithm in OBS networks.
2. TCP over OBS

2.1 Introduction to TCP

TCP sends data in chunks, called segments, which are acknowledged by the receiver. However, a TCP sender has a limit on the outstanding data (the data that has been transmitted and has not been acknowledged yet), which is indicated by the value of the send window. Such limit is in fact the minimum of two limits, one imposed by the receiver, the receive buffer, 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 outstanding data in order not to overload the network. TCP does not initially transmit at full rate, but it uses the slow start mechanism, by which TCP starts probing the network. 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 until a threshold is trespassed. The next phase is congestion avoidance, where the congestion window is increased at a slower rate, at most one segment per round trip time [2].

TCP also provides reliability by detecting the loss of segments and retransmitting. A TCP sender detects the loss of a segment by means of the reception of three duplicate ACKs, or by the triggering of a retransmission timeout. TCP was created with the idea that the loss of a segment is a clear indication of congestion. Thus, when a segment loss is detected, the TCP congestion window is reduced in order to transmit fewer segments, thus lightening the network load and helping to stop the congestion. 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.

2.2 Delayed ACK

The first version of the TCP protocol [3], states that a TCP receiver sends an acknowledgement for each incoming segment. This behaviour was later modified by RFC 1122 [4], 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 should be generated for at least every second full-sized segment and must be generated within 500 ms from the arrival of the first unacknowledged packet. However, this is not the exact behaviour in real-world TCP. The most common implementation is that TCP has a timer that goes off every 200 ms. Thus, the timer can expire anytime within 200 ms since the arrival of an incoming packet. The ACK is therefore sent when a second full-sized segment is received or otherwise when the periodic timer goes off. This is the implementation used, for example, in the Windows TCP stack. Figure 1 shows how this implementation of the delayed ACK algorithm works.

The delayed ACK algorithm was introduced to reduce the load in the network and in the hosts that send and process TCP segments. Due to the cumulative value of the acknowledgements, using delayed ACK has no impact on transmission reliability. Furthermore, it has also been found that using delayed ACK can improve performance in asymmetric networks [5]. However, it has also been demonstrated that delayed ACK reduces performance in certain situations. As described in RFC 2581 [6], TCP increases the congestion window (and so the amount of outstanding data in the network) based on the number of ACKs received. Thus, reducing the
number of ACKs received leads to a slower increase of the amount of sent data. For instance, it has been found that the time spent during the slow start phase is doubled when delayed ACK is used [7]. During the congestion avoidance phase, due to the same reason, the increase of the congestion window is also slower.

![Diagram of ACK behaviour](image)

**Figure 1:** Example of ACK behaviour. (a) Every segment is acknowledged by the receiver, (b) Behaviour of delayed ACK.

### 2.3 TCP over OBS issues

When TCP is used in OBS networks, some OBS features have an impact on TCP performance. The main issues are the burst assembly process in the ingress router, and the contention that occurs in the intermediate nodes when several bursts are competing for the same output port, which usually leads to the loss of a burst due to the lack of practical optical buffers.

Burst losses is the aspect that most influences TCP performance. When a burst is lost, several consecutive TCP segments are lost, and as previously mentioned, when a segment loss is detected, the TCP congestion window is reduced. The effect of the loss of a burst depends on the proportion of the TCP send window that is transmitted in a single burst. If a burst that contains a full window is lost, the TCP sender by no means can be notified of such a loss. All the segments that the sender was allowed to transmit are lost, so, none of the segments arrive at the receiver, which does not send any acknowledgement, and therefore the TCP sender will have to wait for the triggering of the retransmission timeout, which leads to the application of the slow start algorithm. If the lost burst did not contain a full TCP send window, the TCP sender can receive three duplicate ACKs, triggering a congestion avoidance algorithm, which depends on the TCP version employed.

Besides, the burst assembly process introduces an additional delay, which has a two-fold effect. On the one hand, it increases the round trip time, which degrades the TCP end-to-end performance. On the other hand, there is a beneficial correlation effect, called Delayed First Loss gain (DFL Gain), which leads to a performance improvement [8]. Thus, there is a trade-off between these two effects.

The performance of different versions of TCP over OBS networks has been studied by Detti et al. [8], who focused on the Reno version, and by Yu et al., who studied Reno, New Reno and SACK [9], [10]. The latter study showed that SACK is the TCP version that achieves higher throughput. However, none of these studies have
considered the impact of the delayed ACK algorithm in network performance, and that is precisely the aim of this paper.

2.4 TCP on networks with high bandwidth delay product

OBS networks as well as other optical transport networks, suffer from a problem known as the high Bandwidth Delay Product (BDP). TCP is a sliding window protocol, and the amount of data that can be sent at a time can only be at most the size of the send window. The bandwidth delay product is the ‘capacity’ of the link, that is, the number of bits that can be in transit at most. If the TCP send window is smaller than the BDP, there is a waste of the link capacity, and the TCP sender will be idle most of the time. TCP window is limited to $2^{16} - 1$ bytes (64 kbytes), which is much lower than the BDP in most optical networks. In order to achieve higher throughput in this kind of networks, TCP extensions have been defined in RFC 1323 [11], where a TCP scaling option has been introduced, which allows the send window to be as high as $2^{32} - 1$ bytes (4 Gbytes). However, since the congestion window increases very slowly, and decreases abruptly when segments are lost, it is difficult to reach such a send window. Since it is likely that in OBS networks with a high access bandwidth TCP sources will use such option, we have also studied the impact of the delayed ACK algorithm when the window scaling option is employed.

3. Impact of delayed ACK in OBS networks

In this section, we investigate the performance of TCP — with and without the delayed ACK algorithm — over OBS networks by means of simulation, and give qualitative explanations to the results.

Previous TCP over OBS studies have considered that if the access bandwidth is high enough to transmit a whole TCP window in the burst aggregation time, the whole TCP window will be packed and sent in a single burst, and the sender will wait for an acknowledgement before sending new data. If delayed ACK is not used, then every segment is acknowledged by the receiver, and hence, all the acknowledgements will also travel in a single burst. Therefore, the TCP sender will transmit all the new segments in a single burst. This ideal transmission scenario is shown in Fig. 2.a. However, when delayed ACK is used, the last segment of the initial burst may be acknowledged in a random time (within 200 ms), and such an ACK may travel alone in a single burst. This scenario is especially frequent during the slow start phase. Due to this kind of events, the TCP sender does not transmit all the segments in a row like in the ideal case, but a whole TCP window travels in several bursts rather than in a single one. This scenario is shown in Fig. 2.b. Detti [8] found that the DFL gain is proportional to the square of the mean number of segments in a burst; thus, one of the effects of the use of delayed ACK is the reduction of the DFL gain.

Moreover, in OBS networks, contention is a common issue. Thus, bursts may be lost, and hence, the TCP congestion algorithms will reduce the congestion window, and therefore the send window, and may even execute the slow start mechanism. As explained in Section 2.2, when delayed ACK is used, the time spent during the slow start phase is doubled when compared to a scenario without delayed ACK. In the phase of congestion avoidance, the congestion window evolution is also slower with delayed ACK. Consequently, after a burst loss, if delayed ACK is used, it will
take a longer time to reach a value of the congestion window similar to that of the receive buffer. Thus, the higher the receive buffer, the higher the time required to recover from a burst loss using delayed ACK.

![Figures 2(a) and 2(b) showing TCP transmission in OBS with and without delayed ACK.](image)

**Figure 2:** (a) Ideal TCP transmission in OBS: all segments of a TCP send window are packed in a single burst. (b) TCP transmission in OBS with delayed ACK algorithm: the segments of a TCP send window are transmitted in three bursts.

### 3.1 Simulation scenario for TCP over OBS

We have performed a simulation study to analyse the impact of delayed ACK in the throughput of TCP over OBS networks. The simulation model has been implemented in OPNET Modeler 11.0 [12], and is shown in Fig. 3. The model is based on a scenario widely used in the literature [8], [10]. The endpoints, “TCP Client” and “TCP Server” are connected to the OBS edge nodes via a lossless path of 1 Gbit/s of bandwidth and 10 ms of delay. These nodes are directly connected by a fibre link of 10 Gbits/s of bandwidth and 30 ms of delay. The OBS edge nodes implement a timer-based assembly algorithm with a value of 1 ms, which has been shown in previous simulations in literature to be in the range of the optimal values. In every simulation run, 500 files of 10 Mbytes are transferred from the TCP Server to the TCP Client. Throughput is obtained by dividing the file size between the mean time to transfer each of the files. The graphs with the results show the 95% confidence interval. Two ACK strategies are tested, one is to acknowledge every segment and the other is to employ the delayed ACK algorithm.

![Simulation scenario](image)

**Figure 3:** Simulation scenario.

### 3.2 Effects of delayed ACK in Reno and SACK over OBS

The goal of this simulation is to show the effects of the delayed ACK algorithm in two widely used TCP versions, Reno and SACK. Figure 4.a shows the mean throughput for several burst loss probabilities for each version, with and without delayed ACK. Figure 4.b shows the gain obtained when every segment is acknowledged instead of using delayed ACK.
The results confirm that when delayed ACK is not used, a higher throughput is obtained, which is consistent with our previous comments. The results also show that, without delayed ACK, both versions provide the same performance, which is due to the fact that in this scenario every burst loss ends up in the triggering of the retransmission timeout, which is followed by the same behaviour in both versions, the application of the slow start mechanism. Delayed ACK, as we previously explained, makes segments travel in different bursts, so that TCP detects the losses by triple duplicate ACK. In such a case, both TCP versions trigger the congestion avoidance algorithm, but due to selective acknowledgement, which also acknowledges non-consecutive segments, SACK can keep a higher send window than Reno, which only acknowledges consecutive segments. Thus, SACK recovers earlier than Reno from a burst loss. When the loss probability is significant, the gain when every segment is acknowledged is very high, reaching around 60% for SACK and 90% for Reno when the burst loss probability is $10^{-1.5}$. As we explained before, the time to recover from losses is higher when delayed ACK is used; so, the higher the burst loss probability, the higher is the impact of the delayed ACK in throughput. However, when the loss probability is very high the throughput is very low, regardless the use or not of delayed ACK.

3.3 Impact of delayed ACK in SACK and window scaling options over OBS

As it has been commented in Section 2.4, in networks with a high bandwidth delay product, in order to achieve a higher throughput, it is recommended to activate the TCP window scaling option. Thus, it is likely that in OBS networks with a high access bandwidth, TCP sources will use such an option. Consequently, the effect of delayed ACK in TCP over OBS networks should be studied for high receiver buffers. In the current simulation, the receive buffer, which limits the TCP send window, is varied from $2^{16} - 1$ bytes to $2^{22} - 1$ bytes (64 kbytes to 4 Mbytes), and we focus on TCP SACK.

If the window scaling option is used, the receive buffer can be higher than $2^{16} - 1$, so the TCP sender can transmit more data. This behaviour is assessed by the results shown in Fig. 5, where the throughput increases as the TCP receive buffer is set higher. After a burst loss, TCP needs some time to increase the congestion window. The maximum value of the TCP send window is the minimum of the congestion
window and the receive buffer. So, if the receive buffer is higher, it will take a longer time to reach the maximum value. Thus, when delayed ACK is used, the time to reach the maximum value will be even longer. Hence, when the receive buffer is higher, the delayed ACK has a greater impact, as shown in Fig. 6. In the absence of delayed ACK, when window scaling is not enabled (receive buffer of $2^{16} - 1$ bytes), the maximum for the throughput gain is obtained for a loss probability of $10^{-1.5}$ and is practically negligible for loss probabilities lower than that value. However, when TCP window scaling is used, as the buffer is increased, the maximum gain is obtained at lower burst loss probabilities (e.g., $10^{-2}$ for a receive buffer of $2^{22} - 1$ bytes). Although this issue requires further research, we think that the maximum difference in throughput is obtained when the mean time between losses is equal to the mean time that the congestion window needs to reach the maximum value of the receive buffer after a burst loss. So, when the receive buffer is higher, the maximum gain is obtained when the mean time between burst losses is longer, that is, when the burst loss probability is smaller.

![Figure 5: Throughput for SACK when the window scaling option is enabled.](image)

![Figure 6: Throughput gain for SACK with window scaling when delayed ACK is not used.](image)

Besides, when TCP window scaling is used and the receiver buffer is high enough, the absence of delayed ACK has for all burst loss probabilities a significant impact, on the contrary to the case of no window scaling. This is due to the fact that we are measuring the time to transfer 10 Mbyte-files, and with a small receive buffer the time to reach the maximum value of the send window is very small in comparison to
the time to transfer the whole file. In contrast, when the receive buffer is higher, such time is significant in comparison to the time to transfer the whole file.

4. Conclusions

In this paper, we have compared the throughput of TCP over OBS networks with and without the utilisation of the delayed ACK algorithm. The analysis has outlined two penalties when such an algorithm is used, one is a longer time to recover from a burst loss, and the other a reduction of the DFL gain, due to a non ideal transmission by which the segments of a full TCP window are distributed among several bursts.

The use of a strategy to acknowledge every incoming packet instead of using delayed ACK can improve TCP performance, mainly when the window scaling option is used in conjunction with high receive buffer. Therefore, it is highly recommended to activate the window scaling option in OBS networks with a high access bandwidth.

Future work will take into account multiple TCP sources, the addition of synthetic traffic to the TCP sources as well as peer to peer (P2P) traffic. Another work to be done is to consider several TCP flow lengths, from very short flows, like HTTP transfers, to medium/long sized flows, like FTP or HTTP file transfers. Finally, it is interesting to study the algorithm that Allman has proposed in RFC 3465 [13] to improve the throughput when delayed ACK is used.

Acknowledgements

This work has been funded by the Spanish Ministry of Science and Technology under Grant TIC2002-03859, by Junta de Castilla y León under Grant VA037/04, and by the EU IST FP6 Network of Excellence ePhoton-ONe.

References