Burstification effect on the TCP Synchronization and Congestion Window mechanism

Kyriakos G. Vlachos

Computer Engineering and Informatics Department &
Research Academic Computer Technology Institute, University of Patras, Rio, Greece
(Tel: +30 2610 996990, Fax: +30 2610 969 007, email: kvlachos@ceid.upatras.gr)

Abstract—In this paper, we study the burstification effect on the TCP synchronization and TCP congestion window mechanism. It is shown that short assembly times are optimal for flows with similar characteristics, while large assembly time provide a higher notion of fairness. In addition, this paper analyzes the synchronization of multiple TCP flows when aggregated together over the same optical bursts. It is shown that there is a strong TCP synchronization effect upon burst losses, in the sense that flows' windows increase/decrease simultaneously, resulting to a significant variation of outgoing traffic. These deficiencies may be dealt with employing a multi-queue burst assembly scheme with different timers and taking into account the TCP dynamics.

***Index Terms*— Burst Switching, Transport Control Protocol, synchronization, congestion window.**

I. INTRODUCTION

TCP over OBS networks [1] has been studied in previous works [4]-[7] where it has been observed that the burst assembly process at the edge nodes has a significant impact on the end-to-end performance of TCP, mainly because it introduces an unpredictable delay, that challenges the window mechanism used by TCP protocol for congestion control. A useful insight on TCP traffic statistics is given in [8]-[10]. In particular, it was found that short assembly times yield a higher throughput to TCP sources primarily because they reduce the total end-to-end delay associated with the round trip-time delay. However, short assembly time prohibit the fast expansion of the congestion window primarily because sources are allows to transmit only a few segments per round. Long assembly times, are more efficient especially for *fast* TCP flows [8], since they allow the transmission of multiple segments per burst. However, this throughput gain may be canceled by the large burstification delay.

In this paper, we analyze TCP performance over OBS networks and we particularly analyze the burstification effect on the TCP congestion window and on the synchronization of many concurrent flows. We argue that the instant congestion window is a metric to be considered for increasing TCP performance and providing a notion of fairness among the individually aggregated flows. In addition, we analyze the synchronization effect of multiple concurrent flows. This effect has not been in-depth investigated but may lead to temporal overloads in the network. For the detailed investigation of the above mentioned issues, a dedicated TCP-over-OBS simulator was developed using the ns-2 tool. The ns-2 simulator was modified to efficiently manipulate TCP sources and save CPU resources.

The rest of the paper is organized as follows. Section II presents an overview of TCP variants, while Section III discusses the burstification effect on the segments/flow distribution and respectively on the congestion window dynamic. Section IV analyzes the TCP synchronization effect and finally, Section V presents a multi-queue burst assembly scheme that can cope with the abovementioned deficiencies.

II. TRANSPORT CONTROL PROTOCOL VARIANTS

There are a number of TCP versions, combined with a number of different burst assembly schemes. The most interesting are Reno, New Reno and SACK. The main differences among them are the algorithms that they employ when congestion is detected. TCP Reno refers to TCP with *Slow Start*, *Congestion Avoidance*, *Fast Retransmit* and *Fast Recovery* algorithms. When Reno starts, it enters the Slow Start phase first with a congestion window of one segment size and then exponential increase it, upon the acknowledgement of all the packets transmitted. When the window reaches a certain threshold of *w*, it enters, the Congestion Avoidance phase, according to which the windows is now increasing only by one segment after all segments have been acknowledged.

In TCP Reno, there are two kinds of losses identified; the *Time Out* (TO) and *Triple Duplicate* (TD) loss. In the *Triple Duplicate* (TD) case, the sender receives three duplicates ACKs, that acknowledge a new segment, but not the one with the highest sequence number. In that case TCP Reno enters the Fast Retransmit phase, and start transmitting the lost segments. For every successfully transmission of these segments, the sender halves its congestion window by half and receives a TD ACK message for the next lost segment in the burst. In Reno, the maximum number of recoverable segment losses in a congestion window without timeout is limited to one or two segments in most cases. In the case of a *Time Out* (TO) loss case, no ACK is received in a certain time period, denoted by the expiration of a timer. In that case TCP Reno enters the Slot Start phase, and halves its window back to one segment size. New TCP Reno is a slight modification according to which the sender retransmits one lost segment per round-trip-time upon receiving a partial ACK message,

without waiting for a TD ACK and without halving its window until all lost segments are successfully acknowledged. On the other hand, SACK (Selective Acknowledgment) TCP implements a different ACK message, where the non-contiguous set of data that have been received are stored. To this end, the sender is aware of the lost packets and which are transmitted altogether. In that case the congestion window is halved, before linearly increasing again. Detailed SACK performance in OBS networks is clearly superior, as shown in [8], since all the segments that were employed in a burst that was dropped can be identified and subsequently retransmitted at the same round.

TCP's performance (e.g., throughput) depends heavily on burst assembly time due to the extra delay enforced (denoted as burstification delay). Therefore TCP mechanism adjusts its window mechanism upon a burst transmission or reception and thus timer-based assembly schemes may perform better than size-based algorithms. For the timer threshold there could exists an optimal values that maximizes throughput performance in a TCP over OBS network [8]. In [9], it has been shown that optimal performance can also be achieved with an optimal burst length algorithm, while in [5], it is shown that a dynamic assembly algorithm that adjusts the threshold values (e.g., time, burst-length or both) according to traffic statistics can achieve even a better performance. In what follows we have limited our study to TCP-SACK and timer –based burstifiers.

III. Burstification Effect on Flow and Segment Distribution

Although TCP-SACK implementation and timer-based assembly schemes is the most promising combination, very few works exist on providing an in-depth analysis of how segments, flows distribution over the assembled bursts varies with aggregation time. In this section we investigate this and derive conclusions for the congestion window evolution. We have developed a dedicated TCP-over-OBS simulator using ns-2 tool and conducted full-scale experiment on the NSFnet topology that consists of 8 edge and 6 core nodes. Each link was employing a single wavelength at 10Gbps, while access rate was set to 100Mbps equally for all sources. TCP requests were modeled with a Poisson arrival process with a λ=50 flows/sec rate and an exponential inter-arrival time of $1/\mu$=10msec, while TCP file size was modeled with a Pareto distribution process of p load and a min ON size of 40KByte.

Using this set of metrics, it was possible to vary the TCP arrival rate and/or the mean file size, to obtain measurements for different number of simultaneous active TCP flows and different loads. In what follows we have selected a mean file size of 700KB that corresponded to a 2% burst loss ratio and 600-800 active sources for a timer-based burstifier of 1-to 10*msec.* Using these parameters in the simulation experiments, we have measured two basic statistics; the distribution of segments and the distribution of TCP flows over the assembled bursts. Fig. 1 displays the probability density function of (a) the number of segments and (b) the number of different TCP flows per transmitted burst for 1, 5 and 10*msec* burstifier. From Fig. 1, it can be seen that the number of segments as well as the number of flows per transmitted burst

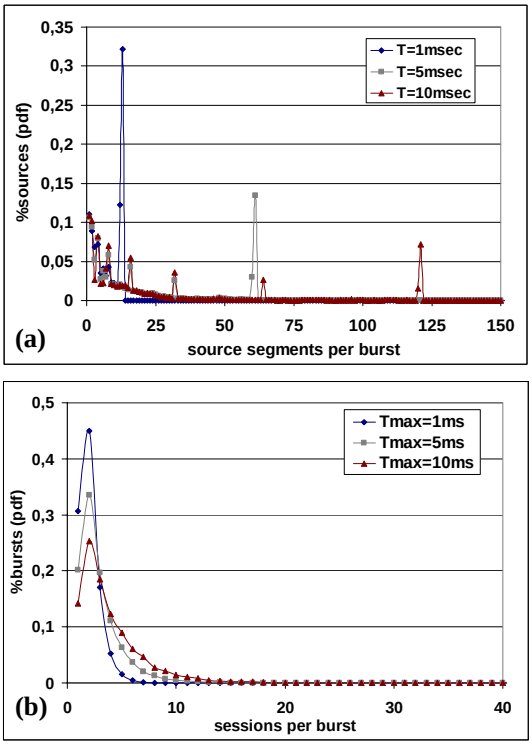
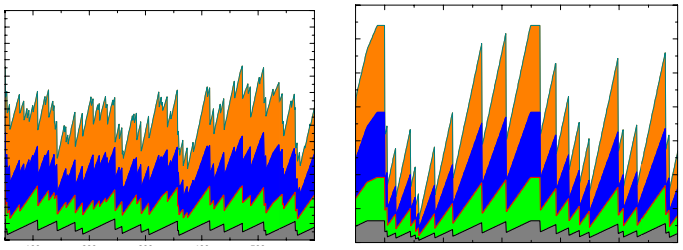


Fig. 1. (a) Segment distribution and (b) number of TCP sources per burst (lost, success) for T_{MAX} = 1, 5, 10msec.

increases with the assembly time. From Fig. 1(b) and in the case of 1*msec* time, it can be seen, that the 80% of the transmitted bursts employ segments from only 2 different sources, while for 5 and 10*msec*, this increases to 4 and 6 respectively. It is clear that large aggregation times contribute in the transfer of a higher number of sources and segments per burst that in turns may lead to smaller completion times, fewer bursts generated but with the trade off that more sources will be potentially affected upon a burst loss.

The pdf distribution of segments per burst (see Fig. 1a) exhibits sharp increases in certain values. These sharp increases are due to the finite access rate in combination with the specific burstification time. For example for 1*msec* assembly time, no more than ~13 segments could be added in a single burst and thus for 1*msec* assembly time, a high percentage of the bursts (~33%) were filled up with 13 segments. For 5*msec* assembly time, this maximum value was found to be 13% of the bursts that were carrying up to 61 segments. Similarly for 10*msec*, this was shifted to 121 segments but for only the 7% of the bursts. The reason that these maximums of the pdf curves were decreasing with the increase of the assembly time can be explained in combination with the congestion window size. It was found out that sources with a large congestion window were capable of adding more segments in the transmitted burst as a result of their higher throughput. To this end, in short assembly times almost all sources could provide bursts with segments and thus the higher percentage of bursts (~33%). In principle however, flows with larger congestion windows have a higher probability to add more segments, since these sources have more unacknowledged segments. This was more evident in large assembly time, where flows with short congestion



(a) **(b)**

Fig. 2. Sum of the bandwidth used by four TCP flows (a) No synchronized TCP flows (b) Synchronized TCP flows

windows stopped sending data traffic, awaiting for an ACK message to arrive after having reached their maximum number of unacknowledged segments. In contrast, flows with larger congestion windows were continuously filling up the bursts with segments. These flows are less in the case of 10*msec* burstifier and thus the small peak in the corresponding pdf curve (~7%). This percentage of flows had a congestion window that was incomparably higher than the assembly time.

As a conclusion, we may argue that short assembly times service less flows, and carry a significant less number of segments. To this end, a high number of flows compete each other to transfer their segments and thus some of them will exhibit a high throughput and some other a poor one. In [11], it was shown that short assembly time provide optimal performance for flows with similar characteristics (same congestion window, file size, etc), yielding to a high variance. Long assembly time provide a higher notion of fairness since they carry more segments from more flows at a time. This smoothes out any traffic instabilities in the sense that individual throughputs are diluted. However the burstification delay that they impose may constrain flow performance (smaller throughput), since sources expand their window at a significant lower rate. Long assembly time provide a throughput gain only when TCP sources have segments to transmit (large congestion windows) otherwise they unnecessarily delay transmission.

In order to truly enhance TCP performance, the instant congestion window is a metric to be considered for determining the optimum assembly time. For example short assembly time should be applied to sources with a relative short congestion window, while larger delays to sources with larger windows. In Section V, such a scheme is presented.

IV. Burstification Effect on TCP Synchronization

A single TCP source transmits *W* bytes, that is the size of its transmission window, per Round Trip Time (RTT). *W* increases as acknowledgements are received, and decreases (halved in practice) when congestion is detected. Thus, *W* and RTT are useful to have an estimation of the bandwidth used by a connection, that is $X(t) = W(t)/RTT$. When several TCP connections share a link, they have to compete for the available bandwidth, that is for a position in the assembling burst. Each connection, as mentioned previously, increases and decreases its bandwidth consumption. Ideally, if connections decrease their window at different moments, a good usage of the bandwidth can be achieved (see Fig. 2a).

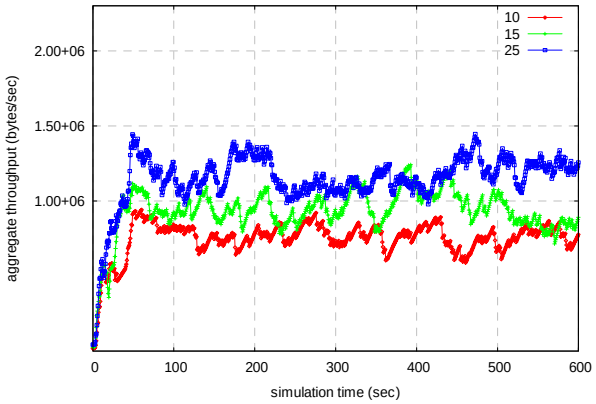


Fig. 3. Aggregated throughput of TCP flows over 600 ms simulation run for per flow assembly, varying the number of TCP flows, N_f.

However, if these moments coincide, the bandwidth usage will resemble a saw-tooth profile (See Fig. 2b), with high peaks and valleys, not using the bandwidth efficiently. Thus, the effect of multiple TCP connections increasing and decreasing the W simultaneously is called TCP Synchronization.

TCP Synchronization is a known problem in packet switched networks. Internet routers are provided with buffers to accommodate temporarily bursts of traffic without having to drop packets. However, when the buffers are full, packets have to be drop, and there is a risk of dropping packets of multiple flows in a row, synchronizing the flows. In the case of packet switched networks, the size of the buffer is also important, since large buffers tend to induce synchronization. Intelligent queue management schemes, like AQM, can help to prevent synchronization. However, it has not been investigated in OBS networks, which are quite different form current packet switched networks, where synchronization also occurs, as we shown below. For example, let us consider there are N_F TCP flows sharing an OBS edge router and all of them have the same RTT. Let us also consider that on average N_{DF} different flows are assembled in the same burst. If *W(t)* is the mean window of the TCP flows, a burst drop will result in the simultaneous reduction of N_{DF} $W(t)/2$ segments in total.

We have studied the burstification effect on the flow synchronization using ns-2 tool and a simple experimental setup. TCP SACK agents were again considered but with CBR traffic sources [12]. A timer-based burst assembler collects the TCP segments and then queues them either in a single burst (*mixed flow* (MF) scheme), or in a separate per flow queue (*per flow* (PF) scheme). The different schemes were considered for studying how multiple flows synchronize in the presence of a burst loss.

The simulations were carried out by keeping the aggregate access bandwidth generated by the N_F TCP agent constant and varying the number of these agents, so that the greater the number of agents the lower the bandwidth of each agent bandwidth. TCP agents start their transmission at random time between 0 and 50s. The evolution of the congestion window is monitored at each TCP agent. The window size is sampled every 0.7 seconds during 600 seconds simulation run

In Fig. 3 and Fig. 4, the aggregated throughput calculated as the sum of the congestion windows of all the sources divided by RTT is presented for N_F =10, 15 and 25 sources in the cases of per flow and mixed flow assembly, respectively. As

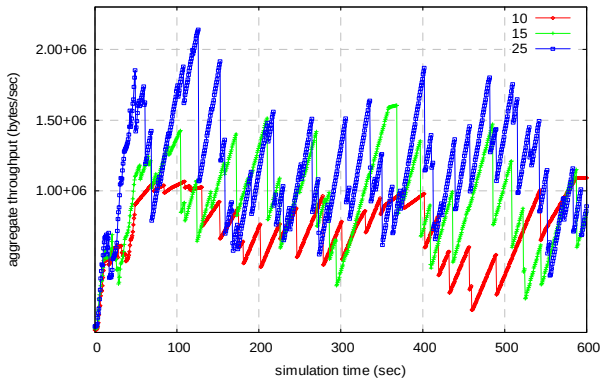


Fig. 4. Aggregated throughput of TCP flows over 600 ms simulation run for mix flow assembly, varying the number of TCP flows, N_f.

the sampled congestion window can be considered as an estimation of aggregate throughput of our scenario. If there were no synchronization, the aggregate throughput of different sources would exhibit a nearly flat profile as shown in Fig. 3 for the per flow strategy. In case of synchronization, the profile is expected to have a saw tooth appearance, as shown in Fig. 4 for mixed flow assembly, where the aggregated sending rate abruptly drops when burst drop occurs. This instability is due to the fact that multiple flows and multiple segments per flow are present in the dropped bursts and cause several agents to decrease the value of their transmission window simultaneously.

Also, when the number of flows increases, the fluctuations are more evident. In fact, as a consequence of the assumption of constant aggregated access bandwidth, the single source rate decreases as the number of flow increases and this involves that each TCP agent is able to send less segments in the same burst. At the same time, during the same assembly time out, the mean of different flows in the same burst was higher.

V. Multi Queue Burst Assembly Scheme

Based on the above analysis, it is clear that a constant timer-based burstifier is not appropriate for TCP over OBS networks. It yields sub-optimal performance and temporarily overloads the network due to flow synchronization. In order to truly enhance TCP performance, we propose a multi-queue burst assembler, where flows are allocated dynamically to different queues. For example each burstifier may employ three different assembly queues with different assembly timers, for sources with small, medium or large congestion windows. The following algorithm is an example of how flow window can be taken into account in the assembly process.

$$T = \begin{cases} 1msec & if\ 1 \leqslant congestion\ \ window < B\ segments \\ 5msec & if\ B \leqslant congestion\ \ window < C\ segments \\ 10msec & if\ C \leqslant congestion\ \ window \end{cases}$$

B,C parameters determine when each flow will move from one to another queue. For example, flows that are in a slow start phase and have a congestion window of less that *B* segments are aggregated together under 1*msec* delay. When their congestion windows reach the limit of *B* segments, then these flows are upgraded to *medium* rate flows and their segments are assembled under a 5*msec* delay. In this way each flow is treated separately and thus upon a burst loss only the flows that will suffer from a data loss will be downgraded to shorter

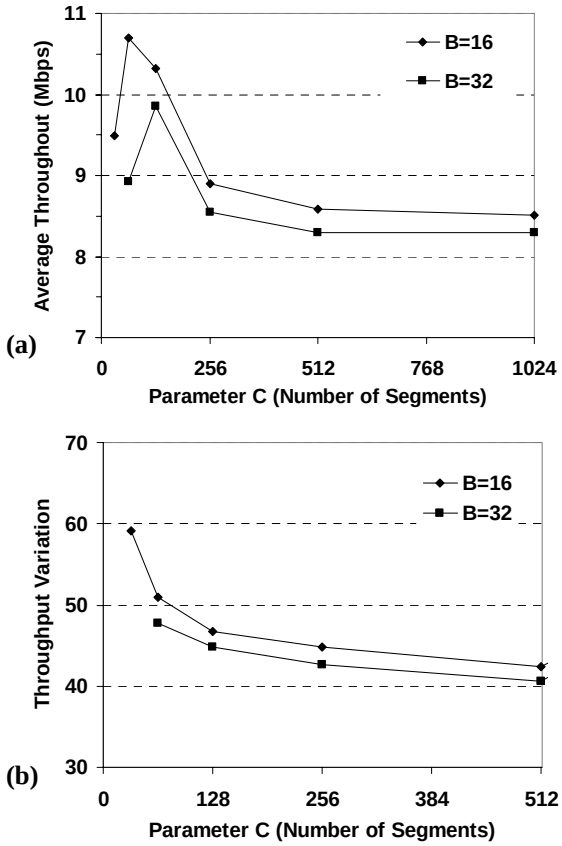


Fig. 5. (a) Average throughput and (b) variance of all flows aggregated at a single node and heading for the same destination for various combination of B and C values.

assembly times. The implementation of this assembly scheme requires the communication of the window size to the burstifier. This can be implemented i.e. by incorporating this information in the TCP header in a future TCP modification.

In order to validate the performance of this scheme, we have measured the yielding throughput (average throughput of all flows at a single edge node) as well as its variance for various *B*, *C* values. As mentioned previously, variance reveals how fair is bandwidth distributed to individual flows, especially in the case that no QoS scheme is applied. Fig. 5a displays the average throughput versus parameter *C,* for two different *B* values; namely for *B=16* and *32*segments. It can be seen that the highest gains in throughout are noted when *C* parameter is fourfold times the *B* value. For example the "*B=16*" curve exhibits a maximum of 10.7Mbps for "*C=64*", while the "*B=32*" curve (corresponding throughput 9.86Mbps) for "*C=128"*. For higher values of *C* (>256), the performance of the system resembles the case of having two only queues. This is because TCP windows do not always reach such a large size due to segment losses. Similarly for higher *B* values (>32), the system performance approximates that of the static case, that is having a single queue with 1*msec* burstifier, while for smaller *B* values (<16), the static cases of 5*msec*.

With respect to Fig. 5 (b), variance is continuously decreasing with the increase of *C* parameter as expected.. It must be noted however, that for the particular case of "*B=16*" and "*C=64*" that exhibited the highest throughput, variation has been significantly decreased to only 51, as opposed to 60 or 70, in

the case of a single queue burstifier with 1 and 5*msec* assembly time [11]. It must be noted here that other individual flows' characteristics such as size, arrival rate may also affect the yielding results, but we may argue that this combination of *B,C* values merges best the performance advantages of long and short assembly times and can support an average throughput of 10.3Mbps with a variance as low as 51.

We have also exploited the multi queue burst assembly scheme to weaken the synchronization effect. In particular, we modified the scheme so that not always the same flows being aggregated together. In other words incoming flows are divided to more than one queue with equal probabilities for the whole assembly period, keeping the flows per queue constant and equal to each other. In the simple case of employing two queues per burstifier and $p_1 = p_2 = 1/2$ is the probability a flow is assigned to any of these two, then the probability of *k* flow to be assigned the first queue is: p_1 if k is even and $1 - p_1$ if k is odd. Similarly the probabilities for the second queue are $1 - p_1$ and p_1 if k is even or odd respectively. Thus, first queue will employ $k/2$ flows if *k* is even (with probability p_1) or $\frac{k+1}{2}$ if *k* is odd.-

The effectiveness of this scheme has been evaluated through simulations using ns-2 tool [12]. Allocation of flows to queues was initiated with the arrival of the first segment of each flow and was kept constant per aggregation cycle. It is then reinstate again. Fig. 6 displays the synchronization effect (number of data on the outgoing link). It can be seen that synchronization has been indeed decreased and we may argue that the random distribution of different flows to different queues desynchronize segment transmission of flows.

In the results shown in Fig. 6, the standard deviation has been reduced by 37% in the case of two-queues and 40% in the case of four queues.

VI. Conclusions

In this paper, we have presented the burstification effect of timer-based algorithms on the synchronization of flows as well as on the TCP congestion window evolution. It has been shown that short assembly time may provide a higher throughput but do not fairly share outgoing capacity between the aggregated flows and thus resulting in a high variance of performance. In contrast long assembly time dilute individual flows and provide a higher notion of fairness. In addition, it has been shown that burst losses in combination with the flow competence for the available bandwidth results in the synchronization of flows' congestion windows. The latter may lead to temporal overloads that cannot be accommodated by the network capacity.

Cleary, it has been shown that a multi-queue burst assembly scheme, can be beneficial for both deficiencies and truly enhance TCP performance. In particular, it is shown that when flows are dynamically assigned to different burst queues synchronization decreases. Similarly, when congestion window is considered as the assembly criterion in a such a multi-queue system, it may truly enhance TCP performance.

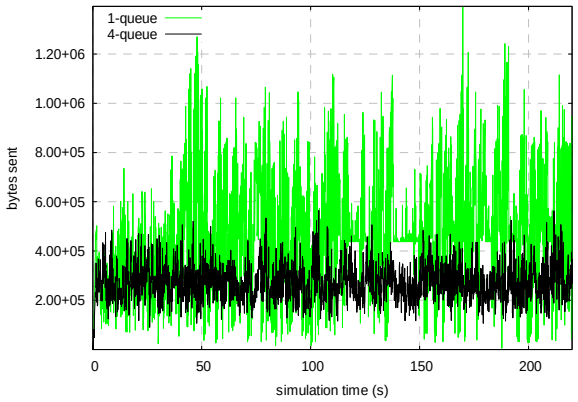


Fig. 6. Flow synchronization employing a dynamic flow allocation algorithm to one and four burst assembly queues.

VII. Acknowledgements

This work has been supported by European Commission through the NoE E-Photon/ONe+ project via the Joint Project on *Optical Burst Switching* (JP-B). The author would like to thank his colleagues *Carla Raffaelli* and *Óscar González de Dios* for their collaboration on this work.

VIII. References

[1] C. Qiao and M. Yoo, "Optical burst switching (OBS)-A new paradigm for an optical internet," J. High Speed Networks, vol. 8, no. 1, pp. 69–84, 1999.

[2] X. Yu, Y. Chen, and C. Qiao, "Study of traffic statistics of assembled burst traffic in optical burst switched networks," in Proc. Opticomm, 2002, pp. 149–159.

[3] Cao, X., Y. Chen, J. Li, and Qiao, C. (2002). IEEE Globecom 2002, 3, 2808 – 2812.

[4] M. Casoni, E. Luppi and M. Merani, "Impact of Assembly Algorithms on End-to-End Performance in Optical Burst Switched Networks with Different QoS Classes". In Proc. of Workshop on Optical Burst Switching (WOBS).

[5] X. Cao, J. Li, Y. Chen, and C. Qiao, "Assembling TCP/IP packets in optical burst switched networks" in Proc. IEEE GLOBECOM, vol. 3, Nov. 2002, pp. 2808–2812.

[6] S. Malik and U. Killat, "Impact of burst aggregation time on performance in optical burst switching networks", in Proc. Optical Network Design and Modelling (ONDM-2005), 2005.

[7] A. Detti and M. Listanti, "Impact of segments aggregation on TCP Reno flows in optical burst switching networks", in Proc. IEEE, INFOCOM 2002.

[8] Xiang Yu et al. "Traffic statistics and performance evaluation in optical burst switched networks", Journal of Lightwave Technology, vol. 22, no. 12, pp. 2722 – 2738, Dec. 2004.

[9] V. Vokkarane, K. Haridoss, and J.P. Jue, "Threshold-based burst assembly policies for QoS support in optical burst-switched networks". In Proceeding of Opticomm, pages 125-136, 2002.

[10] Óscar González de Dios, Ignacio de Miguel, Víctor López, "Performance evaluation of TCP over OBS considering background traffic" in Proceedings of ONDM 2005.

[11] K. Ramantas, K. Vlachos, Ó. González de Dios and C. Raffaelli, "TCP traffic analysis for timer-based burstifiers in OBS networks" in proceeding of ONDM 2007.

[12] Oscar González, Anna Maria Guidotti, Carla Raffaelli, Kostas Ramantas and Kyriakos Vlachos, "On the Synchronization effect of TCP flows in OBS networks", submitted to IEEE Globecom 2007 conference.