

Assembling TCP/IP Packets in Optical Burst Switched Networks

Xiaojun Cao Jikai Li Yang Chen Chunming Qiao

Department of Computer Science and Engineering

State University of New York at Buffalo

Buffalo, NY 14260-2000

Abstract— Optical Burst Switching (OBS) is a promising paradigm for the next-generation Internet infrastructure. In this paper, we study the performance of TCP traffic in OBS networks and in particular, the effect of assembly algorithms on TCP traffic. We describe three assembly algorithms in this paper and compare them using the same TCP traffic input. The results show that the performance of the proposed Adaptive-Assembly-Period (AAP) algorithm is better than that of the Min-BurstLength-Max-Assembly-Period (MBMAP) algorithm and the Fixed-Assembly-Period (FAP) algorithm in terms of goodput and data loss rate. The results also indicate that burst assembly mechanisms affect the behavior of TCP in that the assembled TCP traffic becomes smoother in the short term, and more suitable for transmission in optical networks.

I. INTRODUCTION

TO meet the increasing bandwidth demands and reduce costs, several optical network paradigms have been under intensive research. Of all these paradigms, optical circuit switching (e.g. wavelength routing) is relatively easy to implement but lacks flexibility to cope with the fluctuating traffic and the changing link state; Optical Packet Switching (OPS) is conceptually ideal, but the required optical technologies such as optical buffer and optical logic are too immature for it to happen anytime soon. A new approach called Optical Burst Switching (OBS) that combines the best of optical circuit switching and optical packet switching was proposed by researchers [1] [2], and has received increasing amount of attention from both academia and industry worldwide [3–7].

Within an OBS network, an ingress OBS node assembles IP packets into bursts and sends out a corresponding control packet for each data burst. This control packet is delivered out-of-band and leads the data burst by an offset time. The control packet reserves necessary resources all the way from the ingress node to the egress node where the data burst will be disassembled.

While TCP is the dominant transport protocol today and likely to be adopted in future optical networks, all existing work evaluating the performance of OBS networks assumes that the input traffic is generated by a random generator or at the best a sampled trace file of IP packets, neither of which can be used to evaluate the effects of the congestion control and retransmission by TCP [8,9]. In this paper, we are interested in finding out how the assembly algorithms perform under TCP congestion control mechanisms and what the assembled TCP traffic would be. To this end, we use the TCP traffic as input to study performance of OBS networks and compare different assembly algorithms using NS2 [10]. One of the insights obtained from our study is that TCP performance is more sensitive to the assembly period (AP) than the burst length, and hence ideally, the assembly

period should adapt to the TCP’s sending window size.

The remainder of the paper is organized as follows. Section II briefly introduces the main idea of OBS and describes an existing burst assembly algorithm, then we propose two new assembly algorithms. We present simulation results and analysis in Section III. Section IV concludes our work.

II. BURST ASSEMBLY ALGORITHM IN OPTICAL BURST SWITCHED NETWORKS

Fig. 1 shows the basic procedure of sending one burst from an ingress node to an egress node in an OBS network. At

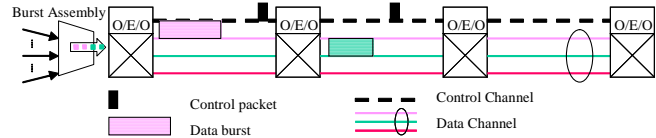


Fig. 1. Optical Burst Switched Networks

the ingress nodes of an OBS network, all TCP/IP packets are assembled into bursts. The ingress node sends out a control (or setup) packet before sending out the data burst. There is an offset time between the control packet and the data burst to give the intermediate OBS nodes enough time to configure their switching fabrics and reserve channel for the following data burst. The control packets are sent out on one or more dedicated control channels (e.g. wavelengths) and go through O/E/O conversion at each intermediate node to provide information about the coming burst. However, the data burst will go through each intermediate node in the optical domain without any O/E/O conversion.

There are many interesting topics in OBS, such as burst scheduling, burst assembly, offset time setting, and contention resolution [4]. The focus of this work is the assembly mechanisms and their interactions with TCP/IP. Currently, how to efficiently assemble IP packets to bursts in an OBS network is still an open issue. In this section, we will describe three burst assembly algorithms.

We assume that the ingress node has one dedicated burst assembly queue for each egress node. All incoming packets will be forwarded to the corresponding queue according to their destinations. When the queue size reaches a threshold or the waiting time of the packets in the queue reaches a threshold, the packets in this queue are sent out as a burst. Fig. 2 illustrates the structure of such a burst assembler.

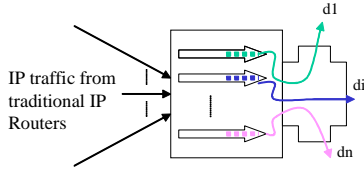


Fig. 2. Burst Assembly

A. The Fixed-Assembly-Period (FAP) Algorithm

A simple and intuitive assembly algorithm is the Fixed-Assembly-Period (FAP) algorithm. The edge OBS node assembles IP packets with the same destination that arrive during a fixed Assembly Period (AP) into a burst and generates a corresponding control packet. The following pseudo code provides a more detailed description of this FAP algorithm, where q_{sd} denotes the assembly queue at ingress node s for destination d and AP_s is the assembly period used by node s .

Event::PacketArrive at node s

d =Destination of the new IP packet;
if (AssembleTimer (q_{sd}) is not running) **then**
 Start a new assembly timer for q_{sd} , $AP_{q_{sd}}$ =initial value;
end if
 Assemble the packet to corresponding queue q_{sd} ;
 Update the burst length information;

Event::AssembleTimerOut (q_{sd})

Fill in and send out a control packet on a control channel;
 Schedule the data burst to be sent out on a data channel after an offset time;
 Stop the assembling timer for q_{sd} ;

Assume that the network uses a static routing algorithm. For the link between two node i and node j , we define a set Q_{ij} , which contains all q_{sd} if and only if the bursts assembled from q_{sd} use the link from node i to node j . Then, for any Q_{ij} , we have the following inequality based on the link capacity.

$$\sum_{q_{sd} \in Q_{ij}} \frac{AvgBL_{q_{sd}}}{AP_s} \leq Channel \times BandWidth \quad (1)$$

where $AvgBL_{q_{sd}}$ denotes the average burst length of queue q_{sd} . $Channel$ is the number of wavelengths on this link and $BandWidth$ is the bandwidth of one channel.

Note that for a specific TCP flow f , in order to prevent TCP from unnecessarily (or prematurely) retransmitting packets in the burst, the assembly period should not be greater than retransmit timeout value (RTO) minus the round trip time value (RTT) associated with flow f [8, 9, 11]. From inequality (1) and above discussion, we have the following constraints on assembly period (Assume that every ingress node uses the same AP.).

$$\frac{\sum_{q_{sd} \in Q_{ij}} AvgBL_{q_{sd}}}{Channel \times BandWidth} \leq AP_s \leq \min_f (RTO - RTT) \quad (2)$$

where, $\min_f (RTO - RTT)$ represents the minimum value of $(RTO - RTT)$ over all TCP flows from s to d . Since $AvgBL_{q_{sd}}$ changes with the value of AP_s , it is difficult to obtain the accurate lower bound of AP_s . One possible approach, however, is to infer $AvgBL_{q_{sd}}$ from the traffic history. More specifically, we may use the following Equation (3), which is similar to the equation for RTT calculation [8, 9, 11].

$$AvgBL_q = \xi \times AvgBL_q + \eta \times SAvgBL_q \quad (3)$$

where $SAvgBL_q$ is the most recently sampled average burst length, $AvgBL_q$ is the smoothed average burst length, and ξ and η are two positive weights ($\xi + \eta = 1$).

Note that there can be many variants of the FAP algorithm. One extreme is to use the same fixed AP value for all ingress nodes (i.e. $AP_s = AP_{s'}$), and another extreme is to use different but fixed AP values for different assembly queues at each ingress node (i.e. $AP_{q_{sd}} \neq AP_{q_{sd'}}$). In the following simulation study, we assume that all the ingress nodes use the same AP value.

B. The Adaptive-Assembly-Period (AAP) Algorithm

As we can see, algorithm FAP is too rigid in that it does not take current traffic situation into account so as to adapt the burst assembly accordingly. This is expected to adversely impact the network performance. Therefore, we propose the Adaptive-Assembly-Period (AAP) algorithm. This algorithm is similar to FAP algorithm, with the difference being that AAP can dynamically change the value of AP of any queue at every ingress node (e.g. $AP_{q_{sd}}$) according to the length of burst recently sent. Given a queue q_{sd} , if we assume that the network uses a single path to route each flow, in the extreme case, the assembled burst traffic from q_{sd} occupies all the output bandwidth. Hence, similar to the rationale behind equation (2), we have following constraints.

$$\frac{AvgBL_{q_{sd}}}{BandWidth \times Channel} \leq AP_{q_{sd}} \leq \min_f (RTO - RTT) \quad (4)$$

where $AP_{q_{sd}}$ is assembly period of queue q_{sd} . We now use the following equation to determine the value of assembly period,

$$AP_{q_{sd}} = \alpha \times \frac{AvgBL_{q_{sd}}}{BandWidth \times Channel} \quad (5)$$

where $\alpha \geq 1$ is called the assembly factor. Note that from equation (1) and (5), we get,

$$\alpha = \frac{BandWidth \times Channel}{\frac{AvgBL_{q_{sd}}}{AP_{q_{sd}}}} \simeq |Q_{ij}| \quad (6)$$

where $|Q_{ij}|$ is the size of Q_{ij} . In other words, Equation (6) indicates that α is the average number of the assembly queues that are sharing the link.

As far as determining $AvgBL_{q_{sd}}$ in equation (5) is concerned, we propose to use Equation (3) by setting $\xi = \frac{1}{4}$ and $\eta = \frac{3}{4}$ in our simulation. The reason for putting more weight on the most recently sampled burst (i.e. $\eta > \frac{1}{2}$) is to allow this assembly algorithm to synchronize with TCP as much as

possible. More specifically, after a long burst is sent out, it is very likely that TCP will still send out more (sometimes twice as many) packets and thus it is better to increase AP. On the other hand, if TCP is done or entering a slow start stage (e.g. a short burst is sent out), it is desirable to reduce the assembly period significantly. In addition, to prevent the burst length from increasing or decreasing by too much, we use lower and upper thresholds (i.e. $\beta > 1$ and $0 < \gamma < 1$) to keep the assembly period within a reasonable range. More details about the AAP algorithm are shown in the following pseudo code.

Event::PacketArrive at node s

d =Destination of the new IP packet;
if (AssembleTimer (q_{sd}) is not running) **then**
 Start a new assembly timer for q_{sd} , ($AP_{q_{sd}}$ =Current (or initial) value);
end if
 Assemble the packet to corresponding queue q_{sd} ;
 Update the burst length information;

Event::AssembleTimerOut (q_{sd})

Fill in and send out the control packet on control channel;
 Schedule the data burst to be sent out on a data channel after an offset time;
 Update the statistical average burst length $AvgBL_{q_{sd}}$ (see equation (3));
 Update the value of Assembly Period for queue q_{sd} (see equation (5));
if ($AP_{q_{sd}} > \beta \times$ current assembly period) **then**
 $AP_{q_{sd}} = \beta \times$ current assembly period;
else if ($AP_{q_{sd}} < \gamma \times$ current assembly period) **then**
 $AP_{q_{sd}} = \gamma \times$ current assembly period;
end if
 Stop the assembling timer for q_{sd} ;

Note that while AAP is much more flexible than FAP, it also involves more complexity (in adjusting the AP) at each ingress node. A compromising approach, called Semi-Adaptive-Assembly-Period (SAP) algorithm is a possible alternative, where all the queues in Fig. 2 at each given node use the same assembly period value, which dynamically changes based on the current traffic. Due to the space limitation, we will not discuss SAP further in this paper.

C. The Min-BurstLength-Max-Assembly-Period (MBMAP) Algorithm

While we do not want to send out data bursts that are too small in order to reduce the overhead, we should send out a burst as soon as possible, and certainly before the first packet in the burst misses its deadline. So we propose the third algorithm, which generates a control packet when a burst exceeds a minimum burst length (MBL) or when the assembly period times out, whichever comes first. These two parameters (maximum assembly period (MAP) and MBL) can be set such that the minimum burst length is smaller than the average burst length (obtained using equation (3)), and the maximum assembly pe-

riod is approximately $\min_f(RTO - RTT)$ (as in equation (2)). The pseudo code for the MBMAP algorithm is as follows.

Event::PacketArrive at node s

d =Destination of the new IP packet;
if (AssembleTimer (q_{sd}) is not running) **then**
 Start a new assembly timer for q_{sd} , MAP=initial value;
end if
 Assemble the packet to corresponding queue q_{sd} ;
 Update the burst length information;
if (AssembledLength \geq MBL) **then**
 Generate burst control packet for this burst;
 Fill in and send out the control packet on a control channel;
 Schedule the data burst to be sent out on a data channel after an offset time;
 Stop the assembling timer for q_{sd} ;
end if

Event::AssembleTimerOut (q_{sd})

Generate a burst control packet for this burst;
 Fill in and send out a control packet on a control channel;
 Schedule the data burst to be sent out on a data channel after an offset time;
 Stop the assembling timer for q_{sd} ;

Note that when we disable the constraint on the minimum burst length¹, this algorithm degenerates into FAP.

III. NUMERICAL RESULTS & PERFORMANCE ANALYSIS

In this section, we study the performance of previously proposed assembly algorithms using TCP traffic as input to an OBS network, as shown in Fig. 3. The italic number on the link in the figure represent the propagation delay (or distance) between two nodes. According to our experiments, there is no significant difference in the performance among different TCP implementations (such as "Reno", "NewReno" or "Sack"). Due to the space limitation, we report only the results obtained using the TCP "NewReno" in this section. We make the following assumptions:

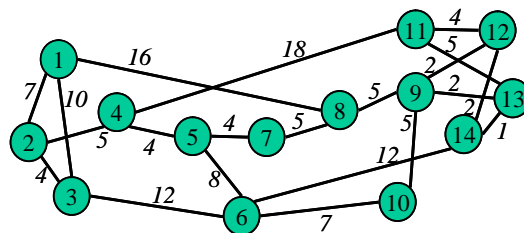


Fig. 3. NSFNet

- Every OBS node has full wavelength conversion capability.
- Each link has bi-directional fibers, each fiber has 8 wavelengths.

¹In following simulation, we denote this by setting MBL=0.

- There is no buffer or FDL in the OBS network.
- The TCP setup and ACK packets go through the control channel.
- FTP connections are used to generate TCP traffic, whose destination is randomly distributed among all the nodes.

A. Comparison Between OBS and OPS

Conceptually, OPS (Optical Packet Switching) is a special case of OBS if the assembly schemes are disabled in OBS (i.e. when $AP=0$).

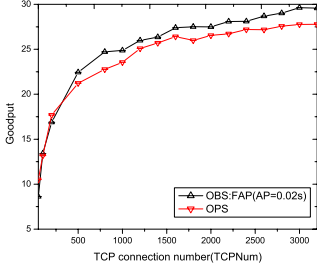


Fig. 4. Goodput ~ TcpNum

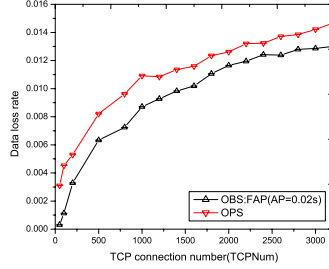


Fig. 5. Loss Rate ~ TcpNum

Fig. 4 and Fig. 5 are the performance comparisons of OBS (using FAP) and OPS in terms of goodput (i.e. the effective data that the destinations receive.) and loss rate as a function of the number of TCP connections. The figures indicate that with an appropriate assembly algorithm and assembly parameters, the OBS networks have better performance than that of OPS networks in that the goodput and data loss rate of OBS with FAP are about 15% better than OPS in most cases. Also note that, as the number of TCP connection increases, the network gets saturated, and the goodput and loss rate tend to be a constant.

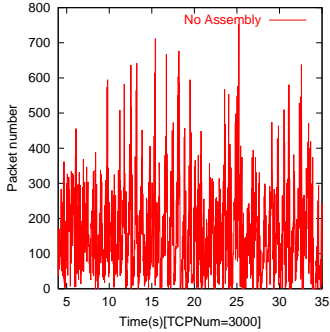


Fig. 6. Packet number ~ Time

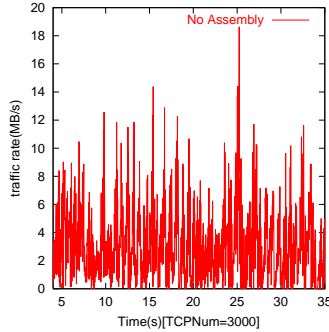


Fig. 7. Data rate ~ Time

In order to explain the above results, we examine the traffic profiles. Fig. 6 ~ Fig. 9 show the difference between original TCP traffic and assembled TCP traffic at a randomly picked node. Fig. 6 and Fig. 7 are the packet number arrival rate and data arrival rate of the original TCP traffic. The number of simultaneously arrived packets may be about 800 in OPS networks. It is very likely that they will cause contentions at the output ports. Similarly, Fig. 8 and Fig. 9 are the results for assembled traffic using FAP algorithm, which shows that burst assembly reduces the variance in the number of bursts/packets

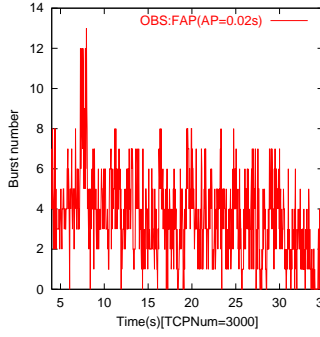


Fig. 8. Burst number ~ Time

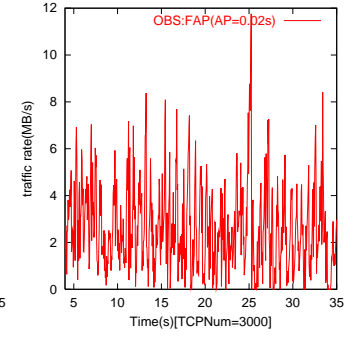


Fig. 9. Data rate ~ Time

as well as the data rate. The smoothing effect is verified by the R/S and variance plot [12] [13] [14] [15] in Fig. 10 and 11, which show the statistical properties of OPS traffic (no assembly) and OBS traffic (after assembly).

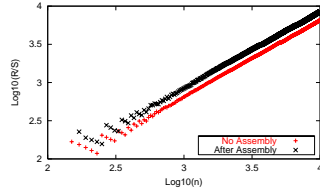


Fig. 10. R/S plot

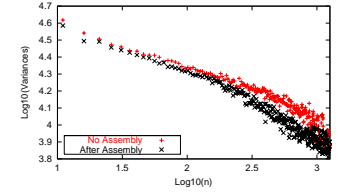


Fig. 11. Variance plot

The slopes for no-assembly TCP traffic and after-assembly traffic are almost the same, which indicates that the self-similarity degrees of both traffic are not much different. In other words, burst assembly algorithm does not change the traffic's long-range dependence property. Fig. 11 exhibits that the variance of assembled traffic is lower than that of the unassembled traffic, indicating that burst assembly indeed changes the traffic's short-term properties. Such a smoothing effect in the short-term makes the assembled TCP traffic more suitable for transmission (i.e. results in a higher goodput and lower loss rate as shown in Fig. 4 and 5).

B. Comparison Between Assembly Algorithms

1) **FAP**: Fig. 12 shows that the performance of FAP is affected by the value of assembly period. More specifically, when the selected assembly period is not appropriate (e.g. > 0.03 when $TCPNum=200$), the goodput of OBS is worse than that of Optical Packet Switching (i.e. $AP=0$). This is because when the assembly period is too large, it takes more time to send packets and receive the corresponding acknowledgement packets, and hence the increase in the TCP sending window size is slowed, which results in a poor performance. This Figure also indicates that the optimal value of assembly period increases with the number of TCP connections. For example, it is about 0.01 and 0.02 when the number of TCP connections increases from 200, 500 to 3000 respectively.

2) **MBMAP**: There are two parameters in Min-BurstLength-Max-Assembly-Period algorithm: minimum burst length (MBL) and maximum assembly period (MAP). Fig. 13 and Fig. 14 are the goodput for MBMAP by varying

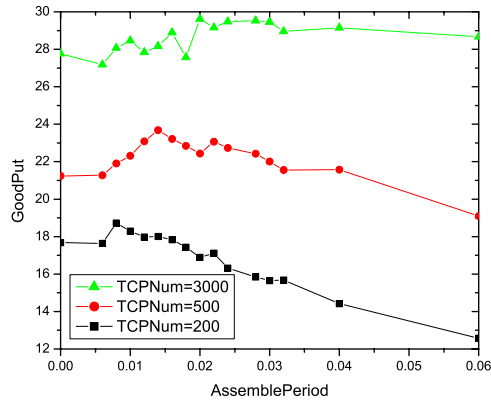


Fig. 12. Goodput ~ Assemble Period

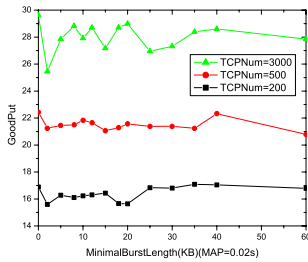


Fig. 13. Goodput ~ Minimum Burst Length

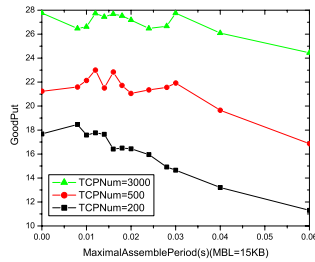


Fig. 14. Goodput ~ Maximum Assembly Period

one of these parameters at a time. More specifically, Fig. 13 indicates how the value of MBL affects the system's goodput, where $MBL=0$ actually represents the FAP algorithm. Fig. 14 shows how the value of maximum assembly period affects the system's goodput. Apparently, the network performance is more sensitive to MAP than MBL because the TCP congestion control and retransmission mechanisms are basically controlled by timers. As can be seen from the figures, for the same assembly period, MBMAP performs as well as FAP in most cases.

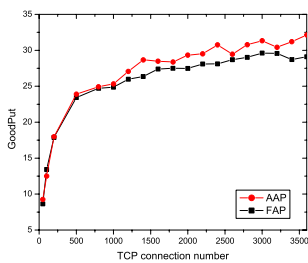


Fig. 15. Goodput ~ TCPNum

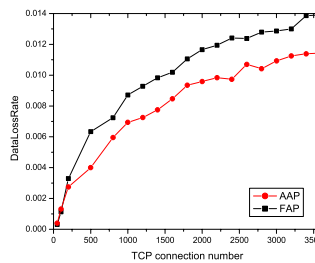


Fig. 16. Loss Rate ~ TCPNum

3) **AAP**: Fig. 15 and Fig. 16 show the simulation results for the AAP algorithm. Here, we set $\alpha = 16$ as the assembly factor (based primarily on the topology and the shortest path routing for every node pair) and set $\beta = 3, \gamma = 1/\beta$. The curves rep-

resenting FAP in these figures are the best performance we can obtain from FAP algorithm by selecting the best (fixed) value of AP. Still, the performance of AAP is better than FAP, especially for a large number of TCP connections. The reason is that AAP adapts the assembly period properly. The ideal situation is that the variation in assembly period can synchronize with TCP's congestion control mechanism, and thus enhances the performance.

IV. CONCLUSION

In this paper, we have proposed and compared three burst assembly algorithms, which are Fixed-Assembly-Period (FAP) algorithm, Adaptive-Assembly-Period (AAP) algorithm and Min-BurstLength-Max-Assembly-Period (MBMAP) algorithm. Based on the properties of TCP flow and bandwidths provisioning in the networks, we have proposed approaches to determine the parameters such as assembly period and the minimum burst length of above algorithms. The simulation results show that AAP is the best, because it adapts assembly periods to match with the TCP congestion control mechanisms. We have used the R/S plot and variance plot to provide valuable insights into the effect of OBS assembly mechanisms on the burstiness and self-similarity degree of the TCP traffic. Our analysis and simulation results have indicated that burst assembly can reduce the simultaneous contention in the core network and make the TCP traffic smoother in the short time scale, and hence improve the performance in terms of goodput and loss rate.

REFERENCES

- [1] M. Yoo, M. Jeong, and C. Qiao, *A high speed protocol for bursty traffic in optical networks*, in SPIE's All-Optical Communication Systems: Architecture, Control and Protocol Issues, Vol. 3230, pp. 79-90, Nov. 1997.
- [2] C. Qiao and M. Yoo, *Optical burst switching (OBS)-A new paradigm for an optical internet*, J. High Speed Networks, Vol. 8, pp. 69-84, 1999.
- [3] J. Turner, *Terabit burst switching*, Journal of High Speed Networks, Vol. 9 No.1, pp. 69-84, 1999.
- [4] Y. Xiong, M. Vandenhouete and H. Cankaya, *Control Architecture in Optical Burst-Switched WDM Networks*, IEEE Journal on Selected Areas in Communications, Vol. 18 No. 10, October 2000, PP. 1838-1851.
- [5] L. Xu, H. Perros, and G. Rouskas, *Techniques for Optical Packet Switching and Optical Burst Switching*, IEEE Comm. Mag., Vol. 39, No. 1, pp. 136-142, Jan. 2001.
- [6] M. Dser and P. Bayvel, *Analysis of wavelength-routed optical burst-switched network performance*, Optical Comm., ECOC 2001, pp. 46-47.
- [7] K. Dolzer, C. Gauger, J. Sph and S. Bodamer, *Evaluation of Reservation Mechanisms for Optical Burst Switching*, Journal of Electronics, Vol. 55, No. 1, pp. 82 - 89, 2001.
- [8] V. Jacobson *Congestion Avoidance and Control*, SIGCOMM Symposium on Communications Architectures and Protocols, pages 314-329, 1988.
- [9] V. Jacobson, *Modified TCP congestion avoidance algorithm*, April 30, 1990, end2end-interest mailing list.
- [10] The Network Simulator — ns-2, <http://www.isi.edu/nsnam/ns/>.
- [11] D.D. Clark and J. Hoe, *Start-up Dynamics of TCP's Congestion Control and Avoidance Schemes*, Technical report, Jun.1995. Presentation to the Internet End-to-End Research Group.
- [12] W. Leland, M. Taqqu, W. Willinger, and D. Wilson, *On the Self-Similar Nature of Ethernet Traffic (Extended Version)*, IEEE/ACM Trans. on Networking, Vol. 2, No.1, PP. 1-15, February 1994.
- [13] F. Avram and M. Taqqu, *Robustness of the R/S statistic for fractional stable noises*, Statistical Inference for Stochastic Processes, pp.69-83, 2000.
- [14] B. B. Mandelbrot, *Limit theorems on the self-normalized range for weakly and strongly dependent processes*, Zeitschrift für Wahrscheinlichkeitstheorie und verwandte Gebiete, 31:271-285, 1975.
- [15] B. B. Mandelbrot and J. R. Wallis, *Computer experiments with fractional Gaussian noises*, Parts 1,2,3 Water Resources Research, 5:228-267, 1969.