

nOBS: an ns2 based simulation tool for performance evaluation of TCP traffic in OBS networks

Guray Gurel¹, Onur Alparslan² and Ezhan Karasan¹

¹*Department of Electrical and Electronics Engineering
Bilkent University
06800 Ankara, Turkey*

²*Graduate School of Information Science and Technology
Osaka University
1-5 Yamadagaoka, Suita, Osaka 560-0871, Japan*

ABSTRACT

Performance evaluation of TCP traffic in OBS networks has been under intensive study, since TCP constitutes the majority of Internet traffic. As a reliable and publicly available simulator, ns2 has been widely used for studying TCP/IP networks; however ns2 lacks many of the components for simulating optical burst switching networks. In this paper, an ns2 based OBS simulation tool (nOBS), which is built for studying burst assembly, scheduling and contention resolution algorithms in OBS networks is presented. The node and link objects in OBS are extended in nOBS for developing optical nodes and optical links. The ingress, core and egress node functionalities are combined into a common optical node architecture, which comprises agents responsible for burstification, routing and scheduling. The effects of burstification parameters, e.g., burstification timeout, burst size and number of burstification buffers per egress node, on TCP performance are investigated using nOBS for different TCP versions and different network topologies.

I. INTRODUCTION

Increasing demand for services with very large bandwidth requirements, e.g., grid networks, facilitates the deployment of optical networking technologies [1]. Using Dense Wavelength Division Multiplexing (DWDM) technology, optical networks are able to meet the huge bandwidth requirements of future Internet Protocol (IP) backbones [2]. Currently, IP routers are interconnected with virtual circuits over synchronous optical networks (SONET) through multiprotocol label switching (MPLS) [3]. However, optical circuit switching (OCS) is not suitable for carrying bursty IP traffic with time-varying bandwidth demand. In addition, delays during connection establishment and release increase the latency especially for services with small holding times. An alternative to OCS is optical packet switching (OPS), which can adapt to changing traffic demands and requires no reservation, but the optical buffering and signal processing technologies have not matured enough for deployment of OPS in core networks in the near future.

Optical Burst Switching (OBS) is proposed as a short-term feasible solution that can combine the strengths and avoid the shortcomings of OCS and OPS [4]. In OBS, the IP packets reaching the edge router are aggregated into bursts before being transmitted in the optical core network. Optical burst switching (OBS) is a sub-wavelength transfer mode that is halfway between optical circuit switching and optical packet switching. OBS separates the data and control planes in the optical and electrical domain, respectively, in order to eliminate the technological problems involved in the all-optical processing of the packet header in optical packet switching. A variable-length optical burst is composed of several IP packets in order to avoid small size optical packets, so that the stringent requirements for transmission and synchronization in the optical domain can be avoided.

nOBS enables performance analysis of OBS networks with wavelength converters and FDLs while carrying TCP traffic, and it implements various burst assembly, scheduling and routing algorithms. nOBS has been developed over ns2 [5].

One of the first ns2 based optical burst switching network simulators is OWns [6], which uses an older version of ns2 and implements a limited number of assembly, scheduling and routing algorithms for OBS networks. OIRC OBS-ns [7] is a re-designed and improved version of OWns. However, OIRC OBS-ns does not allow simulating a network structure composed of OBS subnetworks (clouds) and electronic edge nodes as shown in Fig. 1. Furthermore, OIRC OBS-ns supports only shortest path routing, whereas other routing algorithms cannot be used.

On the other hand, nOBS can be used to simulate general OBS network topologies composed of optical clouds as shown in Fig. 1. nOBS also allows use of any routing algorithm in an OBS network by implementing source routing.

A core OBS network model is shown in Fig. 1 where OBS clouds interconnect edge routers. The edge nodes of an OBS network, i.e., ingress and egress nodes, fulfill the burstification and deburstification functions. The edge node architecture in nOBS allows users to specify the parameters of the burst aggregation algorithm as well as how packets belonging to different TCP flows that are forwarded to the same egress node, are mapped into burstifiers. The edge nodes are also responsible for generating and transmitting the burst control packet, which corresponds to the burst header. The control packet has all the necessary information so that each intermediate optical switch in the core OBS network can schedule the data burst and also configure its switching matrix in order to switch the burst optically. nOBS uses the Just-Enough-Time (JET) reservation protocol [8], where the edge node transmits the optical burst after an offset time following the transmission of the control packet. In JET, the control packet tries to reserve resources for the burst just sufficient enough for transmission of the burst on each link it traverses. The core nodes in nOBS perform the scheduling function using wavelength converters and fiber delay lines (FDL), if necessary. In nOBS, the wavelength converters and FDLs are combined into pools that are shared among all ports. This sharing architecture is called Share-per-Node (SPN), which achieves the best loss performance among other sharing architectures [9]. The user can specify the number of FDLs and wavelength converters in the pools at each node. The scheduling algorithms that are currently implemented in nOBS are Latest Available Unused Channel with Void Filling (LAUC-VF) [10] and Minimum Starting Void (Min-SV) [11]. The routing of the bursts within the OBS network is performed in nOBS using the minimum-hop path between the ingress and egress nodes.

The paper is organized as follows. In the next section, the architecture of nOBS is described in detail. nOBS is used for studying the effects of burstification algorithms and parameters on TCP performance, and the results of this study are presented in Section III. Section IV concludes the paper.

II. SIMULATOR ARCHITECTURE

nOBS is developed by extending existing structures of ns2 (version 2.27). The node and link objects in ns2 are reconfigured with new components to become optical node and optical link. The address classifier at the node entrance has been replaced with a classifier that differentiates TCP segments from optical bursts. The implementation of the optical source routing agent helps realize typical OBS simulation scenarios.

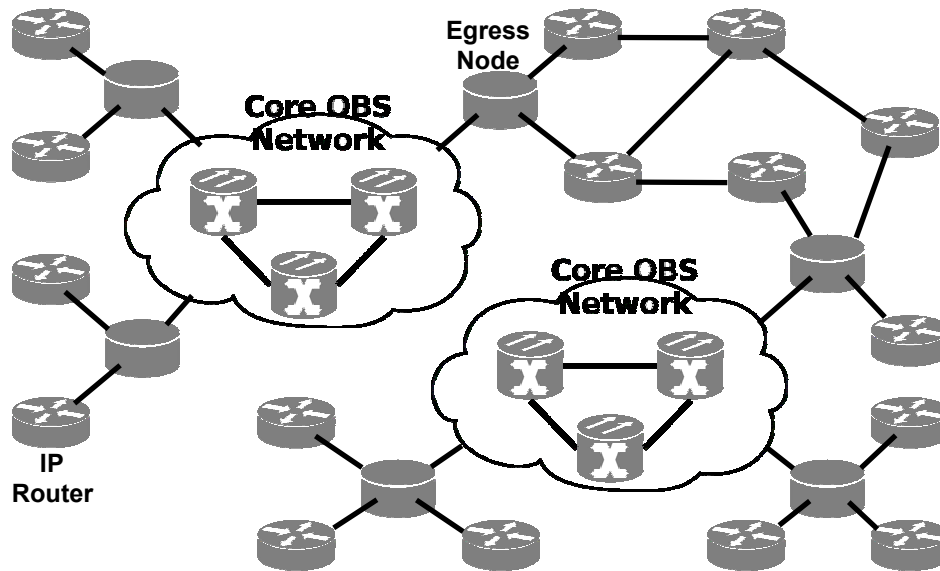


Figure 1. OBS network

The architecture of an OBS node in nOBS is shown in Fig. 2. Ingress, core and egress node functionalities are combined into the nOBS optical node and are indicated by paths 1, 2 and 3 respectively.

The process of burstification (path 1) starts with a packet in electrical domain arriving at the optical node through an access link. This packet is first processed by Optical Classifier (OpClassifier). Upon seeing that the next hop for this packet is in the optical domain, OpClassifier forwards the packet to the Burst Agent (BurstAgent). BurstAgent puts the packet in an assembly buffer that corresponds to a burst and control packet pair. When a burst is ready for

transmission, its associated control packet is sent to OpClassifier and then forwarded to Optical Source Routing Agent (OpSRAgent). OpSRAgent puts the optical domain routing information into the control packet and the corresponding burst. It then checks for a suitable interval through the Burst Scheduler block. This block includes OpSchedule, OpConverterSchedule and OpticalFDLSchedule, which keep records of the reservations on outgoing channels, wavelength converters and FDLs, respectively. If a suitable interval is found, OpSRAgent sends the control packet and schedules the burst to be transmitted after an offset time. Otherwise, the burst is dropped.

OpSRAgent is basically an ns2 source routing agent improved to handle optical packets. When the simulation scenario is described in the TCL code, all nodes (electrical or optical) are commanded to install an OpSRAgent instance and routes for each node to all possible destinations are explicitly defined in the simulation scripts. Therefore, the users can select the routes of packets according to the paths generated by the specific routing algorithm used. In all nodes, newly created packets are sent to OpSRAgent, which writes the path that will be used by the packet in the packet header. In other words, if an application running on ingress router produces data to be sent into the OBS network, the burstification path starts with OpSRAgent, where the route information for the packet is written, followed by the OpClassifier which will forward the packet to the BurstAgent.

Required functionalities of optical nodes are divided into four separate modules (Burst Scheduler, OpSRAgent, OpClassifier, BurstAgent) for reducing the model complexity and allowing easier modification or addition of algorithms. All optical ingress, egress and core nodes require the same functionalities of Burst Scheduler, OpSRAgent and OpClassifier. The only difference between these node types is that core nodes may not need BurstAgent when there is no burstification and deburstification in the core. However, some users may also need to attach traffic agents and burstify/deburstify on the optical core nodes. Therefore, ingress, egress and core nodes share the same node architecture and there is no need to specify the type of the optical node when creating it in the simulation script.

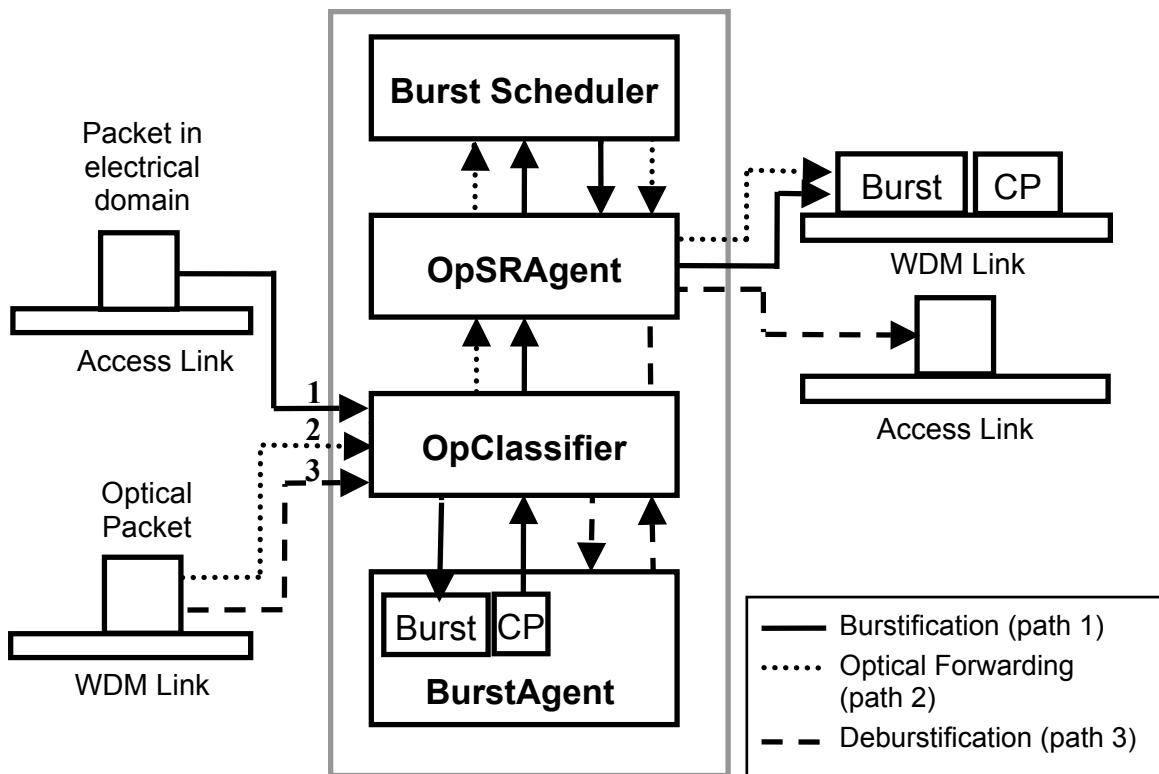


Figure 2. Optical node architecture in nOBS

In the case of optical forwarding (path 2), an optical packet is received by the OpClassifier through an incoming WDM link. Since the next hop is in the optical domain, OpClassifier forwards the packet to the OpSRAgent, which queries the Burst Scheduler block for a valid reservation. If the optical packet is a control packet and a reservation for the associated burst is possible, then the control packet is forwarded to the corresponding WDM link. If the optical packet is a burst and a reservation has been already made, the burst is forwarded to the WDM link. Otherwise, the optical packet is dropped.

When the next hop for an optical packet is not in the optical domain, OpClassifier sends this optical packet to the BurstAgent for deburstification (path 3). If the optical packet is a control packet, it is dropped. If it is a burst, then the packets inside the burst are sent to the OpClassifier, which forwards them to OpSRAgent. OpSRAgent sends these packets through outgoing electrical links towards their destination nodes.

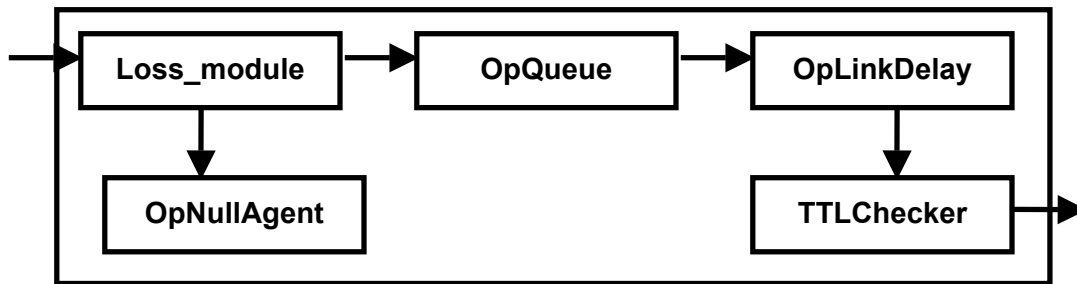


Figure 3. WDM link architecture in nOBS

The architecture of an optical link in nOBS is shown in Fig. 3. This structure is based on the existing ns2 link configuration. Instead of the store-and-forwarding scheme of packet switched networks implemented in ns2, cut-through forwarding is applied. Original Queue of ns2 blocks the link for other packets during the transmission of a packet, until a scheduler event created by LinkDelay signals the end of the packet transmission. The OpQueue module in nOBS immediately forwards all incoming packets to OpLinkDelay without any blocking, packet dropping or queuing since wavelength reservation, contention resolution and FDL buffering operations are already performed by Burst Scheduler and OpSRAgent in the node architecture. It was possible to remove OpQueue and connect loss module and OpDelayLink directly, but OpQueue is kept for easier implementation of future OBS architectures, which may need a queue component on the links. When the Loss module associated with the link determines that an optical packet must be dropped, the packet is sent to OpNullAgent component, which frees individual packets inside the burst. The operations on the link are memoryless and independent of the wavelength. Therefore multiple packets arriving at the same time on different wavelengths can be served without affecting each other.

The main components of nOBS, the classifier, the burst agent, the source routing agent and the optical schedulers, are described below in more detail.

II.1 OpClassifier

A new classifier called OpClassifier is implemented in nOBS for classifying and forwarding packets inside optical nodes. The id numbers of optical nodes in the same domain as this node are given to OpClassifier in a TCL script by using the command *optic_nodes* and stored in a table called *opticonodes*. Therefore, OpClassifier knows the nodes that are in the same OBS domain. When a packet arrives to OpClassifier, OpClassifier checks the type and destination of the incoming packet and handles the packet as follows:

- If the incoming packet is not an optical burst and the packet's destination address is not this node, OpClassifier checks the source routing table of the packet. Looking up in the routing table of the packet, OpClassifier checks whether the packet's next node is in *opticonodes*. If it is, the packet needs to enter the OBS domain, furthermore the node that owns this OpClassifier should act as an ingress node and apply burstification. Therefore, OpClassifier forwards this packet to the burstifier agent called BurstAgent. Otherwise, OpClassifier realizes that this packet is coming from the BurstAgent after the deburstification process. In this case, the packet is leaving the OBS domain, so OpClassifier forwards this packet to the source routing agent that will forward the packet to the next hop over an electronic link.
- If the packet is an optical burst and the packet's destination address is this node, it means that a burst has reached its destination. OpClassifier forwards the packet to the BurstAgent for the deburstification process.
- If the packet is an optical burst and the packet's destination address is not this node, it means that this is a burst in transit. Therefore, OpClassifier forwards this packet to the source routing agent that will forward it to the next hop which is specified in the source routing table of the packet.
- If the packet is not an optical burst and the packet's destination address is this node, it means that the packet is coming from the BurstAgent after deburstification process and the receiver of this packet is in this node. OpClassifier forwards this packet to the port classifier, which will forward the packet to its destination agent.

II.2 BurstAgent

BurstAgent is responsible for the burstification of electronic packets and deburstification of optical bursts. A single BurstAgent is attached to OpClassifier in each optical node. When a new packet arrives from OpClassifier, BurstAgent checks whether this packet is an electronic packet or an optical burst. If the packet received from OpClassifier is an optical burst, BurstAgent disassembles the IP packets inside the payload of the burst and sends these IP packets back to the OpClassifier to be delivered to their destination agents.

If the packet is an electronic packet, BurstAgent compares the source routing table of the packet with the list of nodes contained in the table *optinodes* and finds the corresponding egress node from where this packet will leave the OBS domain. Next, BurstAgent inserts the incoming packet to one of the assembly queues responsible for burstifying packets destined for this destination egress node. The assembly algorithm implemented in the BurstAgent is a hybrid size/timer-based algorithm that keeps track of the size of the burst and the delay experienced by the first packet in the burst. BurstAgent creates a burst when the delay of the first packet reaches a given timeout, or the number of IP packets in the burst reaches a threshold. In our ingress node model, the number of assembly buffers per egress router, M , can be between 1 and the number of flows, N , as shown in Fig. 4. An incoming packet is forwarded to a per egress burstifier queue group based on the routing information, and it is classified further into an assembly buffer based on the flow ID depending on N and M . If an incoming optical packet is the first packet in the assembly queue, BurstAgent starts the burstification delay timer. When the burst is ready for transmission, BurstAgent creates a control packet carrying all the necessary information for this burst. Before sending the burst, BurstAgent copies the packets in the assembly queue to the burst's payload. Then, BurstAgent sends the control packet to OpClassifier. Sending only the control packet to OpClassifier is enough, because other agents in the node can reach the data packet by using a pointer contained in the control packet pointing to the optical burst to be transmitted.

nOBS also allows the user to select whether ACK packets will be burstified or not. Setting *ackdontburst* variable to 1 allows preventing burstification of ACK packets. In this case, ACK packets are sent to the OBS network as soon they are received and they are carried in the OBS network like ghost packets without any dropping or queuing.

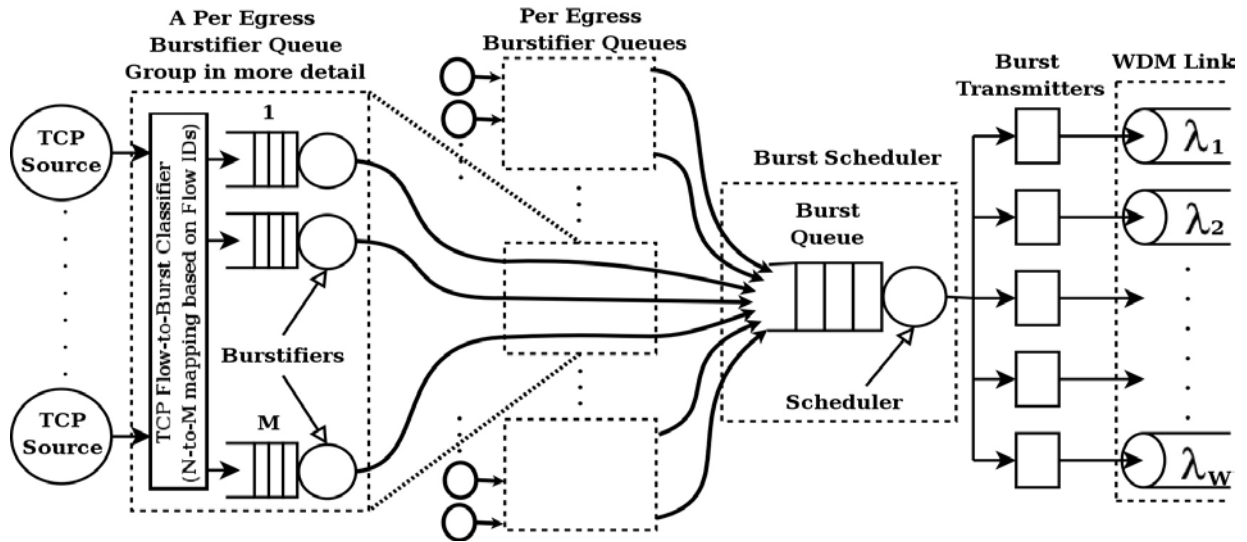


Figure 4. Ingress node model

II.3 OpSRAgent

A new source routing agent called OpSRAgent is implemented in nOBS which is responsible for adding the source routing information to packets, forwarding the packets to links according to the routing information, and controlling when and how to send optical packets using FDLs and wavelength converters. While creating a simulation scenario with nOBS, all the nodes are configured with source routing information within the TCL script. Electrical nodes are configured only with ingress and egress routers of all OBS networks, while optical nodes are informed of routes within the OBS subnetwork they belong to. Using a separate source routing table for optical nodes provides the abstraction, i.e., the cloud structure composed of OBS subnetworks, of the core network within the general topology as shown in Fig. 1.

When OpSRAgent receives a packet, OpSRAgent first checks whether source routing information is available in the packet header and whether this packet is an optical burst or a control packet. If there is no source routing information in the packet header, OpSRAgent considers two scenarios:

1. If this packet is an electronic packet, OpSRAgent writes the routing information to the header of the packet. Then, OpSRAgent checks whether the next hop is an optical node in the same OBS domain. If this is the case, OpSRAgent sends the packet to OpClassifier, which forwards the packet to the BurstAgent for burstification. Otherwise, i.e., if the optical node is the egress node for this packet, OpSRAgent forwards the packet to the next node on an electronic link.
2. If this packet is an optical burst, it means that OpSRAgent has received a newly created burst and control packet pair, so OpSRAgent writes the routing information to the header of both the control packet and the burst.

After ensuring that the source routing information is available in the packet, OpSRAgent checks whether the current node is the destination of this packet. If this is the case, OpSRAgent sends the packet to the OpClassifier. Otherwise, if it is an electronic packet, OpSRAgent sends the packet to the next hop via an electronic link. If this is an optical packet, OpSRAgent tries to send it to an optical link after checking the schedulers. First, OpSRAgent checks the scheduling on this wavelength and link by sending the packet to OpSchedule. OpSchedule returns a result depending on the type of the packet and availability of the channel.

If the packet is a control packet, OpSRAgent takes the following actions based on the result received from the OpSchedule:

1. If there is no contention, OpSRAgent sends the control packet to the optical link for transmission immediately. If this is the first hop of the control packet, OpSRAgent sends the burst corresponding to this control packet to the optical link after delaying the burst for $H\Delta$, where H is the number of hops to be traversed by the burst and Δ is the processing delay per hop.
2. If there is a contention, OpSRAgent checks whether there are unused FDLs or wavelength converters available at the node. If there is, OpSRAgent retries the reservation request, by applying different combinations of available FDLs and converters and chooses the best schedule, if any, according to the scheduling algorithm. OpSchedule learns the availability of FDLs and converters from OpConverterSchedule and OpFDLSchedule, respectively, which are described below. If available FDLs or converters cannot resolve the contention, OpSRAgent drops the control packet.

If the packet is a burst, OpSRAgent takes the following actions based on the result received from the OpSchedule:

1. If there is a reservation for the burst without any contention, OpSRAgent sends the burst to the optical link. If there is a required FDL delay specified in the reservation, OpSRAgent delays the burst before sending to the optical link.
2. If there is no existing reservation for the burst, i.e., the control packet could not succeed in making a reservation for the burst, OpSRAgent drops the burst.

II.4 Optical Schedulers

Each optical node keeps a record of the reservations on outgoing channels, shared FDLs and wavelength converters that are present at the node. OpSchedule holds reservations on outgoing channels while OpConverterSchedule and OpFDLSchedule maintain schedules for wavelength converters and FDLs, respectively. The wavelength converters and FDLs at each node are combined into pools that are shared among all ports at the optical switch, i.e., share-per-node model. The size of the wavelength converter and the FDL pools at each node can be set independently by the user. The user also specifies the maximum FDL delay, which must be limited due to space constraints and for preventing spurious TCP timeouts that degrade the performance significantly [12].

At the ingress node, bursts may be kept in the electrical buffers until they are scheduled and then sent into the optical network. If OpSRAgent cannot find a suitable interval for the burst, it checks possible combinations of wavelength converters and FDLs depending on the node type. If a burst cannot be scheduled, it is dropped. OpSchedule class is responsible for keeping, checking and making reservations on all wavelengths of all links. OpSchedule is connected to the OpSRAgent. When OpSchedule receives an optical packet from the OpSRAgent, it first checks the type of the packet. If the packet is a control packet, OpSchedule tries to do a reservation for the burst specified in the control packet and returns whether reservation is successful or not. If the packet is a burst,

OpSchedule searches for a reservation in its reservation table, which is made earlier by the control packet, and returns whether there is a valid reservation or not. OpSchedule uses Latest Available Unscheduled Channel with Void Filling (LAUC-VF) or Minimum Starting Void (Min-SV) scheduling algorithms in combination with Just Enough Time (JET) signaling. OpSchedule uses a linked-list for storing the reservation list. OpSchedule is responsible for calculating and updating the delay between the control and burst packets.

OpConverterSchedule and OpFDLSchedule are very similar to OpSchedule. These two schedulers are connected to the OpSRAgent, and they are responsible for keeping, checking and making reservations of converters and FDLs at the corresponding nodal pools. They inform the OpSRAgent when OpSRAgent asks for availability in the specified timeline. It is possible to choose whether multiple bursts on a wavelength can use the same FDL subsequently, but the second burst may enter the FDL before the first burst leaves the FDL, by using the *singleburst* parameter from the TCL script. Both schedulers use linked lists for storing the reservations. An important difference between these two schedulers and OpSchedule is that when OpSRAgent sends a control packet to the OpSchedule, if reservation is possible, OpSchedule does the reservation directly. However, OpConverterSchedule and OpFDLSchedule require a parameter called *action*. When a control packet is sent to these schedulers, if *action* variable is set zero, these schedulers only return whether reservation of converter or FDL is possible. They do not do the reservation, unless *action* variable is set one. This is because the scheduling algorithm may use a combination of FDL and wavelength conversion for resolving the contention, and the OpSRAgent must make sure that both the queried FDL and converter are available. If both schedulers return an affirmative reservation signal, then OpSRAgent informs the schedulers to perform the actual reservations.

In the next section, we present some numerical results for the burstification of TCP traffic that are obtained using nOBS.

III. SIMULATION RESULTS

The numerical study presented in this paper analyzes the effects of the burst assembly architecture and parameters on the performance of TCP flows. In the first part of the simulations, the core network is simply modeled as a single fiber with Bernoulli distributed drop probability p , with 1 Gbps bandwidth and 10ms propagation delay as shown in Fig. 5. The access links have 155Mbps bandwidth each with 1ms link propagation delay.

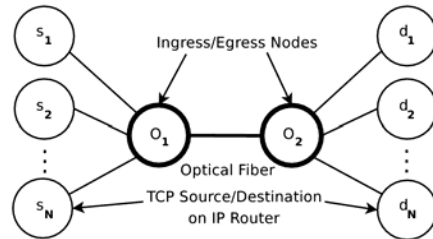


Figure 5 Topology used in simulations

The total goodput for $N=10$ TCP Newreno flows with $p=0.01$ and $M=10$ is shown in Fig. 6 for a range of assembly timeouts and burst size thresholds with a hybrid size/timer-based burst aggregation algorithm. The MSS of the TCP sources are set to 1040 bytes. Receive windows of destination nodes are set to 10000 MSS to avoid the limitation on congestion window increase. In order to study the effect of the burstification timeout on TCP performance, we used the feature of nOBS which allows that TCP ACK packets are not burstified.

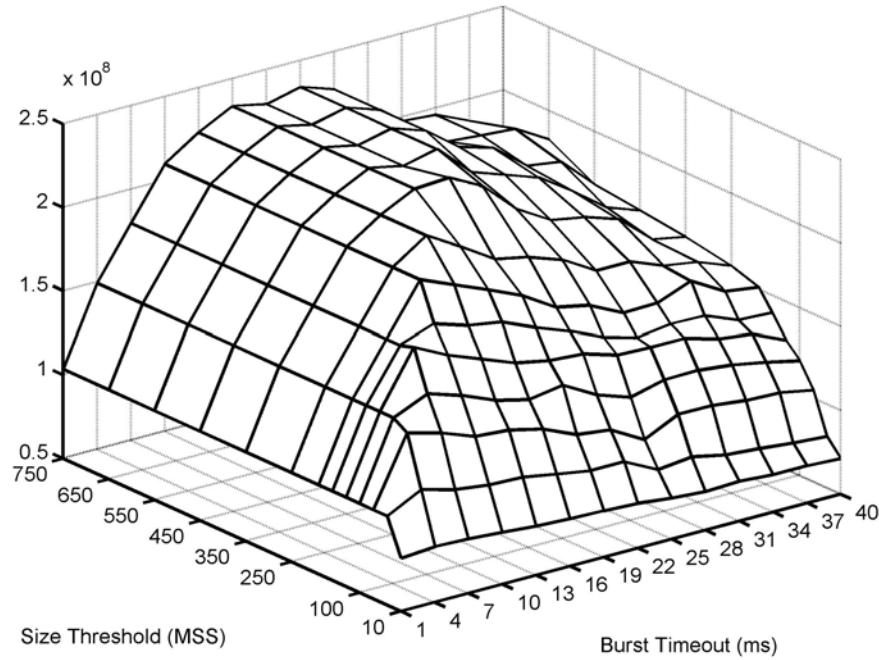


Figure 6. Total goodput achieved with hybrid size/timer-based burst assembly algorithm ($p=0.01$)

For a fixed timeout, the figure shows that goodput increases as the size threshold is increased. However, when the burst size threshold becomes larger than the maximum achievable burst size determined by the current timeout, size/timer-based algorithm reduces to timer-based algorithm and goodput does not change. For the largest burst size threshold, the algorithm acts as the timer-based burstifier. On the other hand, since size/timer-based algorithm becomes size based for a timeout value of infinity, the values with the largest timeout gives an idea about the performance of the size-based algorithm.

For a fixed burst size threshold, the achievable burst size increases with increasing timeout until the current size threshold is reached. We observe that goodput improves with increasing timeout in this region. The effects of high time correlation between delivery and loss events of consecutive packets from a TCP flow as a result of statistically independent burst losses are noted as *correlation benefits* [13]. An important effect of this correlation is the increase in the number of packets sent by a TCP source before noticing a loss event as the burst size increases and this behavior yields larger congestion windows and higher throughput (*Delayed First Loss (DFL) gain*) [14]. That is why, for a fixed burst size threshold, the goodput improves as the achievable burst size rises up to the size threshold with increasing timeout. Once the size threshold is reached, however, further increase in the timeout leads to performance deterioration due to additional burst assembly delay (called the *delay penalty* [13], [14], [15], [16]).

Since the timer-based algorithm achieves the highest goodput, we resort to the timer-based burstification in the rest of the paper for studying the effect of the number of the burstifiers on TCP performance. The TCP Newreno performance over a range of time threshold values are plotted in Fig. 7 and Fig. 8 for $p=0.001$ and $p=0.01$, respectively, for different values of the number of burstifiers, M . The remarks that we made for Fig. 6 can be observed also in Fig.7 and Fig. 8. The figures show that as the assembly time threshold is increased, goodput first increases, then starts to decrease. In the region where goodput increases with timeout, the delay penalty is small and the DFL gain is dominant, therefore increasing the burst size increases the goodput. On the other hand, the improvement provided by the DFL gain saturates after some time threshold value and the delay penalty begins to dominate which causes the goodput to deteriorate.

Fig. 7 and Fig. 8 also demonstrate the significant effect of number of burstifiers on TCP goodput. We observe that the TCP goodput increases for all timeout values as M increases. TCP flows sharing an aggregation buffer are affected from successful delivery and burst loss events together and thus have a tendency to become synchronized. When a burst is lost, TCP flows that have packets in that burst decrease their congestion windows simultaneously. This quick reduction of accessible optical bandwidth results in serious performance degradation. Increasing the

number of burstifiers per egress node, we can decrease the level of synchronization between TCP flows and obtain higher bandwidth utilization as seen in the plots.

Another important observation is that the rate of deterioration in goodput as time threshold is increased depends on loss probability p . When p is large, the congestion window cannot increase to large values due to more frequent burst losses. In this case, the increase in time threshold does not increase the burst size significantly and the increase in DFL gain with increasing time threshold is not significant. As a result, the goodput decreases more rapidly with increasing time threshold due to the delay penalty. On the other hand, large bursts are generated as the time threshold is increased for small p , and the DFL gain increases with increasing time threshold. This partially compensates the effect of the delay penalty, and the goodput does not degrade much with increasing time threshold.

TCP flows are classified as *slow* when only one of their packets is found in a given burst, *fast* when their whole congestion window is found in the burst and *medium* otherwise [13]. The results presented up to this point deal with medium flows since there is not an upper limit on the congestion window sizes. In order to see the performance of fast flows and examine the effect of number of TCP flows better, the network is simulated with $N=100$ Newreno flows with the receive windows set to 128 MSS. Fig. 9 shows the results of the simulations for $p = 0.01$. As the size of the sender's congestion window cannot exceed the receiver's window, DFL gain stays constant at its maximum for large timeouts. Consequently, the effect of delay penalty on goodput can be seen more clearly for large values of timeout and the goodput decreases more rapidly with increasing time threshold. In addition, it is observed that a relatively low number of buffers may perform close to the per-flow aggregation case. From Fig. 9, we observe that for larger values of N , smaller number of burstifiers is sufficient to obtain performances relatively close to the per-flow burstification, i.e., $M=N$. Since the cost of additional burstifiers can be compromised by the improvement in goodput, employing moderate number of buffers with respect to the number of flows constitutes a cost-effective solution.

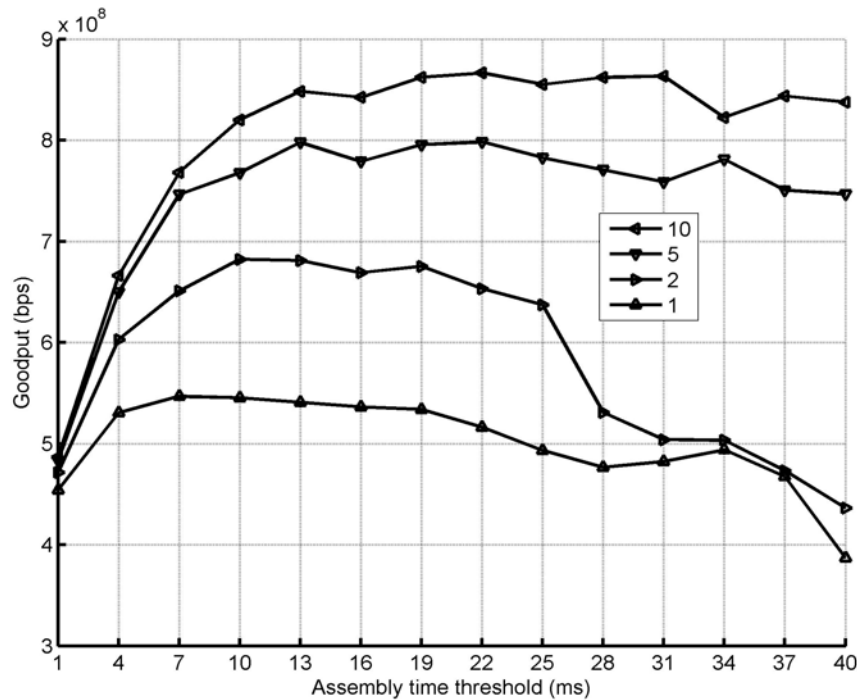


Figure 7. Total goodput with timer-based assembly for $N = 10$, $p = 0.001$, $M = 1, 2, 5, 10$ and Newreno TCP

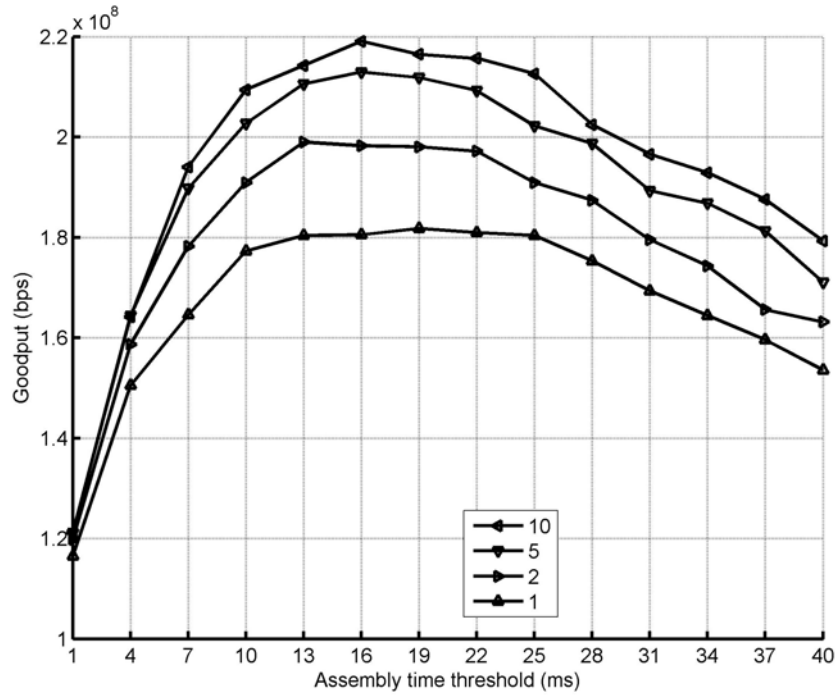


Figure 8. Total goodput with timer-based assembly for $N = 10$, $p = 0.01$, $M = 1, 2, 5, 10$ and Newreno TCP

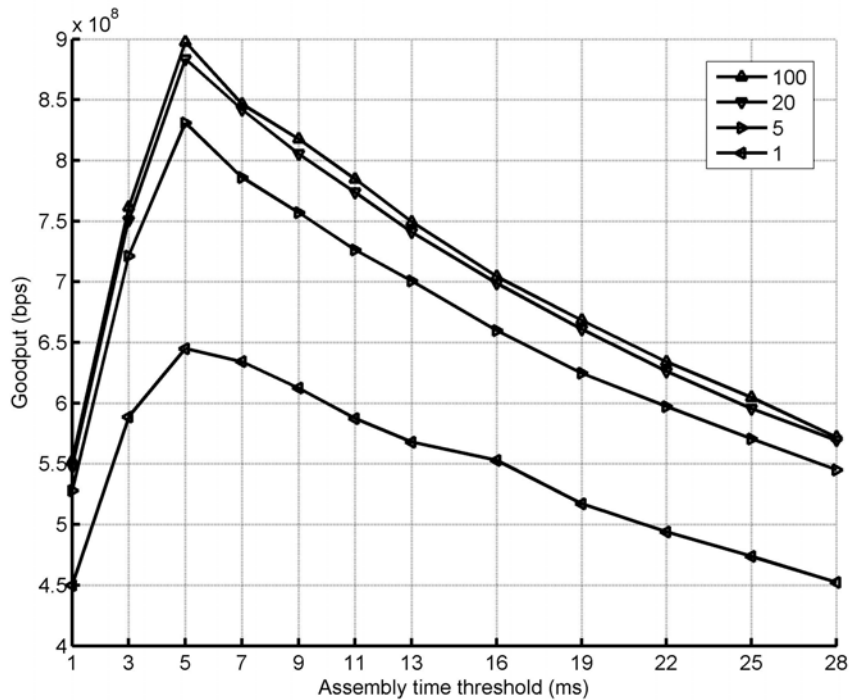


Figure 9. Total goodput with timer-based assembly for $N = 100$, $p = 0.01$, $rcv_wnd = 128$ MSS, $M = 1, 5, 20, 100$ and Newreno TCP

In order to emphasize the role of number of burstifiers, we computed the optimum timeout for $p=0.01$ and $p=0.001$ reaching the maximum goodput for three TCP versions, Newreno, Reno and Sack, for different number of burstifiers, $M=1, 2, 5$ and 10 for $N = 10$ and $M=1, 5, 20$ and 100 for $N = 100$. Fig. 10 and Fig. 11 show the performance of the timer-based algorithm with $N=10$ for $p=0.001$ and $p=0.01$, respectively. Similarly, Fig. 12 and

Fig. 13 show TCP performance for $N = 100$. We observe that increasing the number of burst assemblers significantly improves the goodput for all three TCP versions since synchronization between TCP flows is reduced as the number of burstifiers is increased.

We also observe that although all three TCP versions exhibit similar characteristics as the number of burstifiers is changed, TCP Sack achieves the highest goodput among the three TCP versions. Sack outperforms the other two versions since it quickly retransmits the lost segments with selective acknowledgements. Reno and Newreno have very close performances, however Newreno slightly outperforms Reno.

In the second part of the simulations, nOBS is used to simulate an OBS network composed of 3 ingress, 3 egress and 9 core nodes. The topology of this OBS network is shown in Fig. 14. In this topology, there are 10 TCP connections between each source-destination pair S_i-D_i , $i = 1, 2, \dots, 9$, i.e., the total number of TCP connections carried over the OBS network is 90. Each of the core links has a capacity of 1 Gbps and a propagation delay of 2.5 ms. Each ingress-optical switch and optical switch-egress link has a capacity of 1 Gbps and a propagation delay of 0.1 ms. On the other hand, each access link, i.e., interconnecting source and destination nodes to ingress and egress nodes, has a capacity of 500 Mbps and negligible propagation delay.

The routing table used in these simulations is given below for each source-destination pair (S_i-D_i , $i = 1, \dots, 9$), where I_k denotes ingress node k , OS_l denotes optical switch l and E_m denotes egress node m .

```

S1 → I1 → OS1 → OS2 → OS3 → E1 → D1
S2 → I1 → OS1 → OS2 → OS5 → OS6 → E2 → D2
S3 → I1 → OS1 → OS2 → OS5 → OS8 → OS9 → E3 → D3
S4 → I2 → OS4 → OS1 → OS2 → OS3 → E1 → D4
S5 → I2 → OS4 → OS5 → OS6 → E2 → D5
S6 → I2 → OS4 → OS5 → OS6 → OS9 → E3 → D6
S7 → I3 → OS7 → OS4 → OS5 → OS6 → OS3 → E1 → D7
S8 → I3 → OS7 → OS4 → OS5 → OS6 → E2 → D8
S9 → I3 → OS7 → OS8 → OS9 → E3 → D9

```

Similar to the first part of the simulation studies, flows destined for each egress node are burstified using $M = 1, 2, 5, 10$ burstifiers per egress node in order to investigate the effect of the number of burstifiers on TCP Newreno performance. For each value of M , the burst assembly time threshold is varied in order to find the maximum goodput achieved by each TCP connection. The average maximum goodput, which is computed by taking average over all TCP connections, is shown in Fig. 15 for each value of M . The maximum average goodput increases by more than 20% as the number of burstifiers per egress node is increased from $M = 1$ to 2. Further increase of M from 2 to 10 generates an additional goodput increase of less than 1%. Since increasing the number of burstifiers at the ingress node results in an increase in cost and complexity, switches with efficient burst assembly and relatively low cost and complexity can be built by considering moderate values of the number of burstifiers.

IV. CONCLUSIONS

nOBS, an ns2 based network simulator for the performance evaluation of TCP over OBS networks is presented. We used nOBS in this paper in order to study the effects of the number of burstifiers used at the edge routers on TCP performance. Simulations show that increasing the number of assemblers per destination reduces the negative effects of synchronization between TCP flows occurring as a result of burst losses. We show that TCP goodput is increased significantly when edge routers with multiple burstifiers per destination are used, and the goodput increases as the number of burstifiers increase. This conclusion holds for different TCP versions, different number of flows, different network topologies and different loss probabilities.

ACKNOWLEDGEMENTS

This work is partially supported by FP6 IST e-Photon/ONE+ NoE project and by the Scientific and Technological Research Council of Turkey (TUBITAK) under project 104E047. The authors also would like to thank to Ismail Cirak who has performed some of the simulations in this paper.

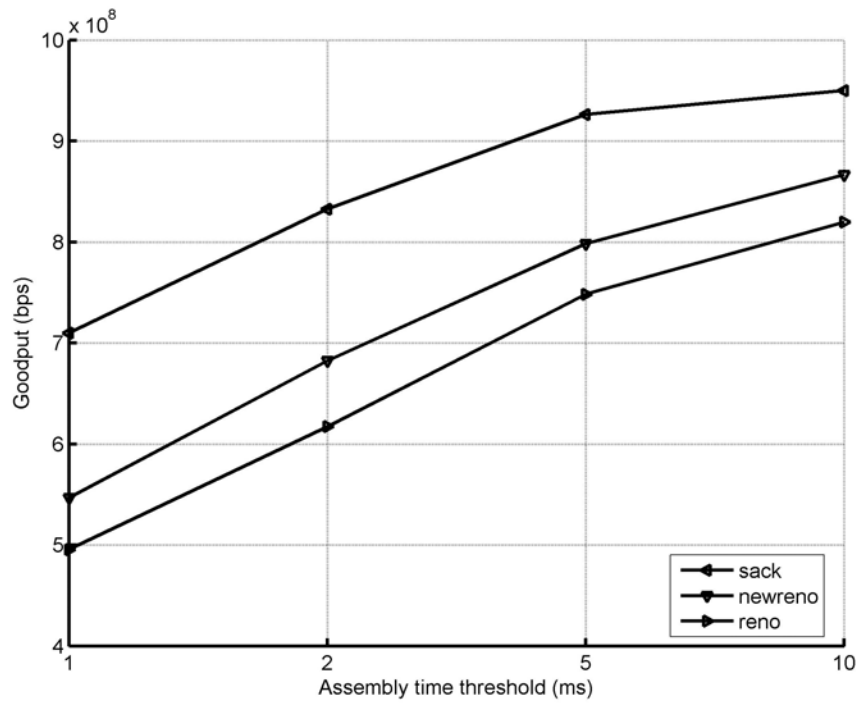


Figure 10. Maximum goodput as a function of number of burstifiers for $p = 0.001, N=10$

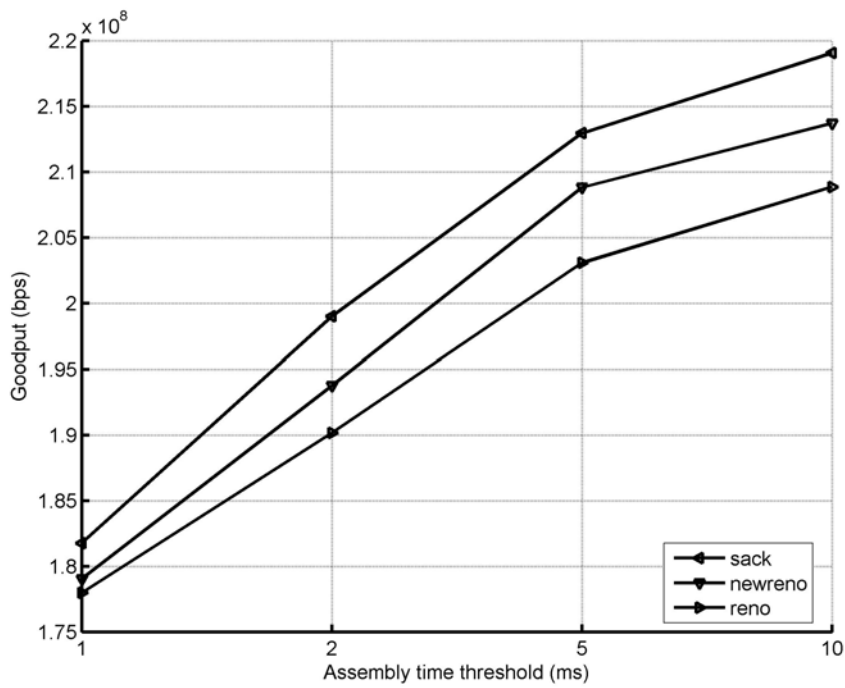


Figure 11. Maximum goodput as a function of number of burstifiers for $p = 0.01, N=10$

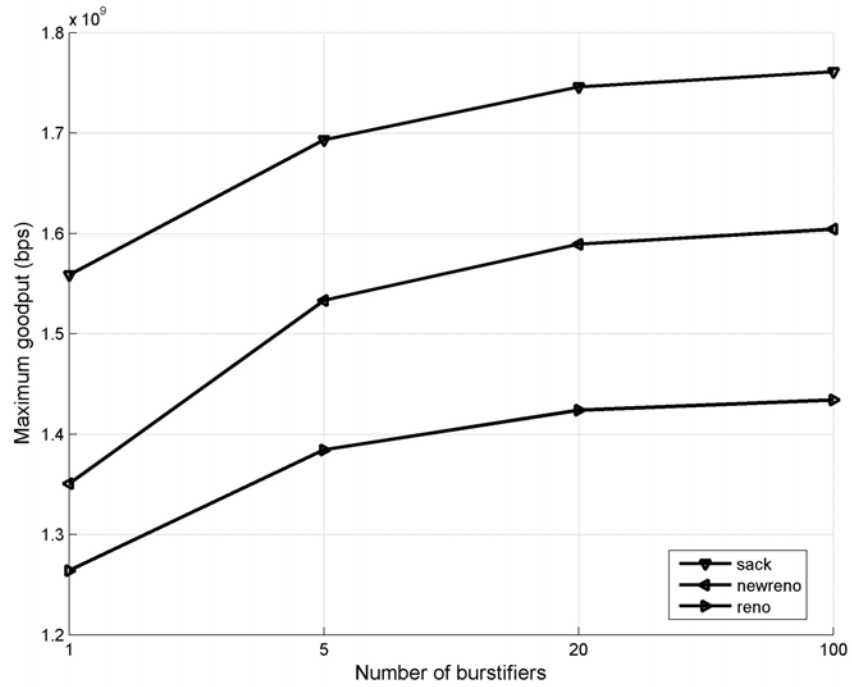


Figure 12. Maximum goodput as a function of number of burstifiers for $p = 0.001, N=100$

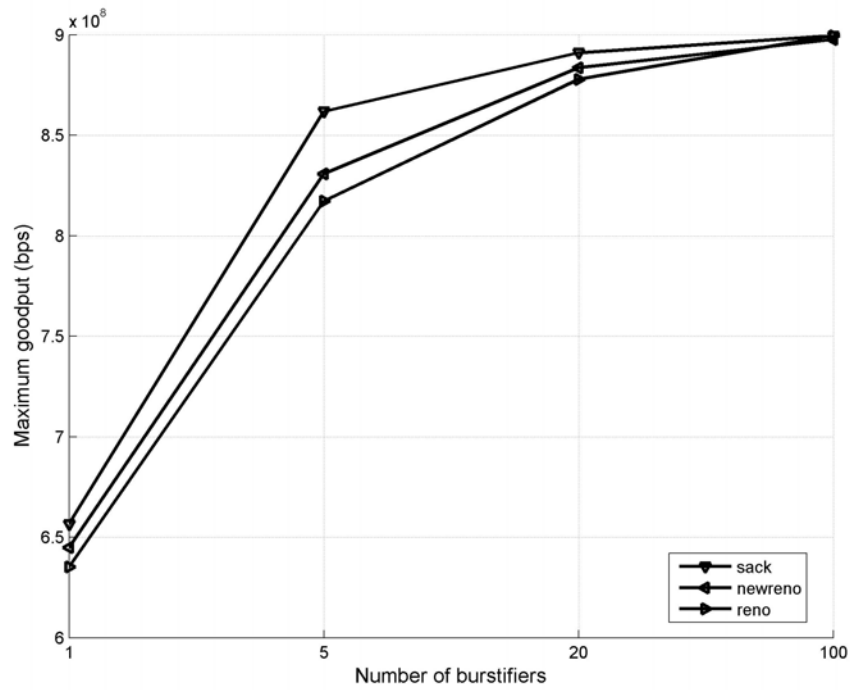


Figure 13. Maximum goodput as a function of number of burstifiers for $p = 0.01, N=100$

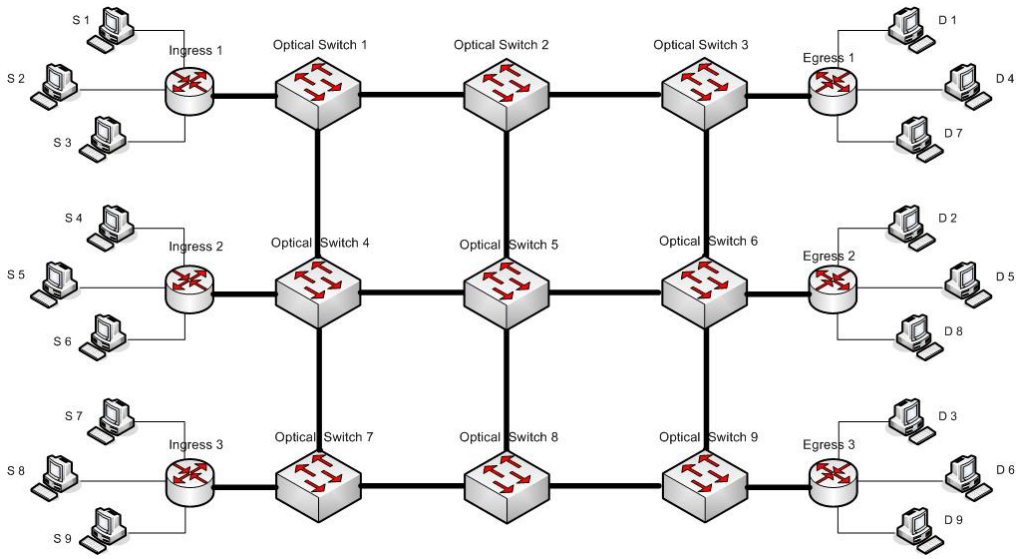


Figure 14. OBS core network topology

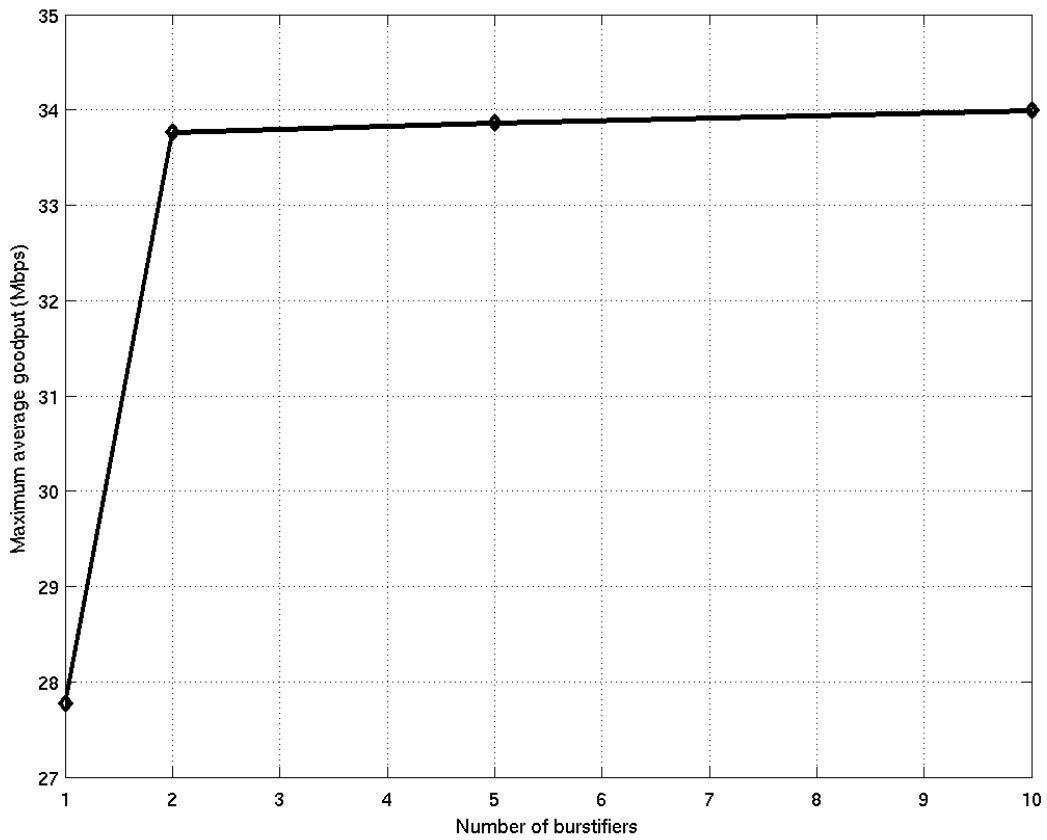


Figure 15. Maximum average goodput as a function of number of burstifiers

REFERENCES

- [1] LISTANTI (M.), ERAMO (V.), SABELLA (R.), Architectural And Technological Issues For Future Optical Internet Networks, *IEEE Communications Magazine*, 38, no. 9, pp. 82-92, Sep. 2000.
- [2] TURNER (J.), Terabit Burst Switching, *J. High Speed Networks*, 8, no 1, pp. 3-16, 1999.
- [3] YAO (S.), XUE (F.), MUKHERJEE (B.), YOO (S.), DIXIT (S.), Electrical Ingress Buffering And Traffic Aggregation For Optical Packet Switching And Their Effect On TCP-Level Performance in Optical Mesh Networks, *IEEE Communications Magazine*, 40, no 9, pp. 66-72, Sep. 2002.
- [4] YOO (M.), QIAO (C.), Optical Burst Switching (OBS)- A New Paradigm for an Optical Internet, *Journal of High-Speed Networks*, 8, no 1, pp. 69-84, 1999.
- [5] The Network Simulator –ns-2, developed by L. Berkeley Network Laboratory and University of California Berkeley, <http://www.isi.edu/nsnam/ns>.
- [6] WEN (B.), BHIDE (N.), SHENAI (R.), SIVALINGAM (K.), Optical Wavelength Division Multiplexing (WDM) Network Simulator (Owens): Architecture and Performance Studies, *Optical Networks Magazine*, 2, no 5, pp. 16–26, Sep. /Oct. 2001.
- [7] JEONG (H.), MO (H.), KANG (M.), OBS-ns: an NS-2 based Optical Burst Switching (OBS) Simulator, *Proceedings of 9th Communications and Networking Simulation Symposium*, 2-6 April 2006.
- [8] YOO (M.), QIAO (C.), Just-Enough-Time (JET): A High Speed Protocol for Bursty Traffic in Optical Networks, *IEEE/LEOS Summer Topical Meetings*, pp. 26–27, Aug. 1997.
- [9] ERAMO (V.), LISTANTI (M.), PACIFICI (P.), A Comparison Study on the Wavelength Converters Number Needed in Synchronous and Asynchronous All-Optical Switching Architectures, *IEEE Journal of Lightwave Technology*, 21, no. 2, pp. 340-355, Feb. 2003.
- [10] XIONG (Y.), VANDENHOUTE (M.), CANKAYA (H.), Control Architecture In Optical Burst-Switched WDM Networks, *IEEE Journal on Selected Areas in Communications*, 18, no 10, pp. 1838–1851, Oct. 2000.
- [11] XU (J.), QIAO (C.), LI (J.), XU (G.), Efficient Channel Scheduling Algorithms In Optical Burst Switched Networks, *Proceedings of IEEE Infocom '03*, 3, pp. 2268 - 2278, 30 March-3 April 2003.
- [12] YU (X.), QIAO (C.), LIU (Y.), TCP Implementations and False Time Out Detection in OBS Networks, *Proc. of IEEE INFOCOM'04*, 2, pp. 774–784, 7-11 March 2004.
- [13] DETTI (A.), LISTANTI (M.), Impact Of Segments Aggregation On TCP Reno Flows In Optical Burst Switching Networks, *Proc. IEEE INFOCOM'02*, 3, pp. 1803-1812, 23-27 June 2002.
- [14] YU (X.), LI (J.), CAO (X.), CHEN (Y.), QIAO (C.), Traffic Statistics and Performance Evaluation In Optical Burst Switched Networks, *IEEE/OSA Journal of Lightwave Technology*, 22, 12, pp. 2722- 2738, Dec. 2004.
- [15] YU (X.), QIAO (C.), LIU (Y.), TOWSLEY (D.), Performance Evaluation Of TCP Implementations In OBS Networks, *Tech. Rep. 2003-13*, CSE Dept., SUNY, Buffalo, NY, 2003.
- [16] GOWDA (S.), SHENAI (R.), SIVALINGAM (K.), CANKAYA (H.), Performance Evaluation Of TCP Over Optical Burst-Switched (OBS) WDM Networks, *Proc. IEEE ICC'03*, 2, pp. 1433-1437, 11-15 May 2003.