CONGESTION WINDOW BASED ADAPTIVE BURST ASSEMBLY FOR TCP TRAFFIC IN OPTICAL BURST SWITCHING NETWORKS

A THESIS

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL AND

ELECTRONICS ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCES

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Seçkin Özsaraç

September 2008

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Ezhan Karaşan(Supervisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Nail Akar

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assist. Prof. Dr. İbrahim Körpeoğlu

Approved for the Institute of Engineering and Sciences:

Prof. Dr. Mehmet Baray
Director of Institute of Engineering and Sciences

ABSTRACT

CONGESTION WINDOW BASED ADAPTIVE BURST ASSEMBLY FOR TCP TRAFFIC IN OPTICAL BURST SWITCHING NETWORKS

Seçkin Özsaraç
M.S. in Electrical and Electronics Engineering
Supervisor: Assoc. Prof. Dr. Ezhan Karaşan
September 2008

Burst assembly is one of the key factors affecting the TCP performance in Optical Burst Switching (OBS) networks. Timer based burst assembly algorithm generates bursts independent of the rate of TCP flows. When TCP congestion window is small, the fixed-delay burst assembler waits unnecessarily long, which increases the end-to-end delay and decreases the TCP goodput. On the other hand, when TCP congestion window becomes larger, the fixed-delay burst assembler may unnecessarily generate a large number of small-sized bursts, which increases the overhead and decreases the correlation gain, resulting in a reduction in the TCP goodput. Using simulations, we show that the usage of the congestion window (*cwnd*) size of TCP flows in the burst assembly algorithm consistently improves the TCP goodput (by up to 38.4%) compared with the fixed-delay timer based assembly even when the timer based assembler uses the optimum assembly period threshold value. One limitation of this proposed method is the assumption that the exact value of the congestion window is available at the burst assembler. We then extend the adaptive burstification algorithm such that the burst assembler uses estimated values of the congestion window that are obtained via

passive measurements at the ingress node. It is shown through simulations that even when estimated values are used, TCP goodput can achieve values close to the results obtained by using exact values of the congestion window.

Keywords: Optical Burst Switching, Adaptive Burst Assembly, TCP over OBS, Congestion Window Estimation

ÖZET

OPTİK ÇOĞUŞMA AĞLARINDAKİ TCP TRAFİKLERİ İÇİN SIKIŞIKLIK PENCERESİ TABANLI UYARLAMALI ÇOĞUŞMA OLUŞTURMA ALGORİTMASI

Seçkin Özsaraç
Elektrik ve Elektronik Mühendisliği Bölümü Yüksek Lisans
Tez Yöneticisi: Doç. Dr. Ezhan Karaşan
Eylül 2008

Çoğuşma oluşturma, Optik Çoğuşma Anahtarlama (OBS) ağlarındaki TCP performansını etkileyen önemli faktörlerden birisidir. Zamanlayıcı tabanlı çoğuşma oluşturma algoritması, TCP akışlarının hızından bağımsız olarak çoğuşmaları üretir. TCP'nin sıkışıklık penceresi küçükken, sabit-gecikmeli çoğuşma oluşturucu gereksiz yere uzun bekler ki; bu da uçtan uca gecikmeyi artırır ve TCP'nin performansını düşürür. Öte yandan TCP'nin sıkışıklık penceresi büyükken, sabit-gecikmeli çoğuşma oluşturucu gereksiz yere fazla sayıda küçük-boyutlu çoğuşmalar üretir ki; bu da başlık yükünü artırır ve ilinti kazanımını düşürüp TCP performansında azalmaya sebep olur. TCP akışlarının sıkışıklık penceresi (*cwnd*) boyutunun çoğuşma oluşturma algoritmasında kullanılmasının, sabit-gecikmeli zamanlayıcı tabanlı oluşturmaya göre hatta zamanlayıcı tabanlı oluşturucu optimum oluşturma periyodu eşik değerini kullansa bile, TCP performansını mütemadiyen artırdığını (%38.4'e kadar) simülasyonlar ile gösterdik. Önerilen bu metodun bir kısıtlaması sıkışıklık penceresinin kesin değerinin çoğuşma oluşturucusunda mevcut olduğu varsayımıdır. Bundan dolayı uyarlanır oluşturum algoritmasını; çoğuşma oluşturucunun, sıkışıklık penceresinin ağ giriş

düğümlerinde pasif ölçümler vasıtasıyla elde edilen kestirilmiş değerlerini kullandığı şekilde genişlettik. Simülasyonlar vasıtasıyla gösterilmiştir ki; TCP perfomansı, kestirilmiş değerler kullanılsa bile sıkışıklık penceresinin kesin değerleri kullanılarak elde edilen sonuçlara yakın değerlere ulaşabilmektedir.

Anahtar Kelimeler: Optik Çoğuşma Anahtarlama, Uyarlanır Çoğuşma Oluşturma, OBS Üzerinde TCP, Sıkışıklık Penceresi Kestirimi

ACKNOWLEDGMENTS

I gratefully thank my supervisor Assoc. Prof. Dr. Ezhan Karaşan for his endless support and supervision throughout the development of this thesis. I would also like to thank Assoc. Prof. Dr. Nail Akar due to his great guidance related to my academic life.

I also cannot deny the contribution of my colleagues to my thesis. Thank you for your assistance and friendship.

Contents

List of Figures

List of Tables

To My Family ...

Chapter 1

INTRODUCTION

The bandwidth usage in the Internet is doubling every six to twelve months [1]. To cope with the exponentially growing traffic demand, fiber-optic transmission is the most promising physical medium to meet the demand. Optical transmission technology is able to transfer signals to longer distances with greater reliability than the copper wiring technology. With the recent developments in the fiber technology, 40 Gbits/s capacity is achieved for a single wavelength in a fiber. Moreover by the usage of Wavelength Division Multiplexing (WDM), up to 160 wavelength channels in a single fiber can be supported by multiplexing operation. As a result, several Tbits/s capacity is achievable over a single fiber [2].

Currently, WDM networks are used as the major backbone links for long distance carriers who use Synchronous Optical Network (SONET) as the standard interface [3]. However, in the current deployment of wavelength division multiplexing (WDM) optical networks, optical-to-electrical-to-optical (OEO) conversion is required at each step since WDM networks operate on point-to-point links. On the other hand, the ultimate goal of optical networks is a network without any OEO conversion, which is called an All Optical Network (AON).

To realize AONs, several different architectures have been proposed. Among these, Optical Circuit Switching (OCS) networks, which is also called Wavelength Routed Networks (WRNs), is the first one that has been proposed. In OCS networks, circuit connections, that are called *lightpaths*, are set between the source and destination nodes. The lightpaths are established with the connection setup request messages and the positive/negative replies (ACK/NACK) according to the availability of the resources. Once a wavelength is reserved for a connection, it is not available until the teardown message, which terminates the lightpath between the source and destination nodes. As a result, the connection establishment process requires a delay of at least a round trip time [4]. In WRNs all the user data travels entirely in optical domain. Only the control packets are processed in the electrical domain. Thus OEO conversion process is substantially reduced, where in the limiting regime of infinitely long duration connections, OCS networks converge to AONs. However, OCS networks are not able to support frequently changing user traffic because of its quasi-static nature [5].

Another proposal to realize AONs is the Optical Packet Switching (OPS) networks, which are conceptually ideal. In OPS networks, an optical packet contains both the payload and the optical packet header. While the payload remains in the optical domain throughout its journey in the OPS network, the optical packet header is processed in the electrical domain. Therefore in each node, the payload waits for the processing of the header to finish and this may result in the generation of queues in the OPS network nodes. As a result, optical buffers are needed to store the optical packets because of the store-and-forward structure of the OPS networks. However, current unavailability of optical buffers is the most significant bottleneck of OPS network technology [6, 7]. Although the Fiber Delay Lines (FDL) may behave like optical buffers, they provide limited and pre-determined delays.

Recently, the Optical Burst Switching (OBS) networks have been proposed [8]. In OBS networks, an ingress OBS node assembles the user data packets from its clients and generates various size units, called *bursts*. Then a Burst Control Packet (BCP), which include the header information is sent for each burst [7, 9]. Thus the ratio of the control packets to the user data packets is less than or equal to one, whereas this ratio is equal to one for OPS networks. Moreover, BCPs are sent on a dedicated control channel, which consists of one or several wavelengths on a fiber. Therefore, the control plane and the data plane in OBS networks are separated in the electrical and optical domain respectively to eliminate the technological problems involved in the realization of OPS networks. Also with the usage of appropriate reservation protocols, which is explained in Chapter 2, the need for optical buffers are prevented in OBS networks. As a result, bursts traverse the OBS network in a cut-through manner rather than the store-and-forward structure of the OPS networks.

The burst assembly algorithm running on the ingress nodes have a great impact on the sizes and the inter-arrival time statistics of the bursts. Hence, due to the variance in the transmission duration of the bursts, OBS networks lie between the OCS and OPS networks. For instance when the optical burst contains only one user data packet, the behavior of OBS converges to the OPS networks. When the optical burst contains infinitely many user data packets, OBS converges to the OCS networks. Therefore, OBS networks combine the advantages of both circuit and packet switching networks. As a consequence, with respect to the existing optical technology, OBS paradigm is the best choice among the other switching paradigms; namely WDM optical networks, OCS and OPS networks. The above discussions about OCS, OPS and OBS networks are summarized in Table 1.1.

Our focus in this thesis is mainly on the burst assembly algorithms especially with TCP traffic flows. This is because the burst assembly algorithm is an

Table 1.1: Comparison of the optical switching paradigms

Optical Switching Paradigm	Bandwidth Utilization	Optical Buffer Requirement	Adaptivity to Bursty Traffic	Processing Overhead	Connection Setup Latency
OCS	Low	No	No	Low	High
OPS	High	Yes	Yes	High	Low
OBS	High	No	Yes	Low	Low

important factor on the performance of OBS networks. The key parameters in the burst assembly algorithm are the maximum burst size, minimum burst size and the pre-set assembly timer threshold value [10]. In the literature, there are variants of the burst assembly algorithms, which can be classified into four categories: *timer based*, *burst length based*, *mixed timer/burst length based* and *adaptive* burst assembly algorithms. The burst assembly algorithms are explained in detail in Chapter 2.

When TCP flows over OBS networks are considered, the TCP performance is significantly affected by the burst assembly algorithm. To be specific, the burst assembly mechanism introduces a delay penalty in TCP throughput since the round-trip times of the TCP flows are increased. This is simply because the incoming packets to the ingress node wait in the burstifier queue until the optical burst is generated before traversing the OBS network. On the other hand, the enlargement of the transmission units from single packets to bursts increase the TCP performance, which is so-called *correlation gain* [9, 11, 12, 13, 14].

In this thesis, we propose a new adaptive burst assembly algorithm for TCP traffic in OBS networks, which is called *congestion window (cwnd) based burst assembly algorithm (CWBA)*. In this algorithm, *cwnd* size of the TCP flow is taken into account in determining when to form the burst, where the assembled burst sizes are directly proportional to the *cwnd* size of TCP. In addition, CWBA does not have any pre-set variables unlike the other adaptive burst assembly algorithms in the literature. Therefore, CWBA has the capability to work on

any network topology without the need of any adjustment for optimality. Besides, the delay penalty introduced by the burst assembly process is minimized since the delay penalty in CWBA is only equal to *cwnd* packets' transmission time without any extra delay, which is not the case in the other assembly algorithms.

The minimization of the delay penalty by CWBA algorithm is verified through simulations performed in nOBS [15], which is an ns2 [16] based simulation tool for OBS networks. Gurel et al. [17] has shown that the timer based burst assembler performs as well as mixed timer/burst length based burst assembler and better than burst length based assembler as far as TCP goodput is concerned. Thus, in our simulations we used the timer based burst assembly algorithm as the reference point. In the end, results show that the usage of CWBA algorithm is always beneficial and CWBA's goodput performance is better than the timer based assembler by up to 38.4%. Moreover, in [18] it is shown that the flavor of TCP have an impact on the throughput in OBS networks. Especially, TCP SACK has the highest throughput among TCP Reno, New-Reno and SACK when the timer based assembler is used. It is also shown that the performance of CWBA is the same for TCP Reno, New-Reno and SACK algorithms.

In addition to our first proposal, we also propose a hybrid version of the CWBA algorithm and the timer based burst assembly algorithm, which is the *mixed cwnd/timer based burst assembly (MCWBA)* algorithm. The crucial property of MCWBA algorithm is that it puts an upper bound on the delay introduced by the burst assembly process. The performances of the two burst assembly algorithms are investigated again with the simulations that are performed in two different network topologies. MCWBA algorithm performs as well as CWBA algorithm. In the first simulation topology, MCWBA performs 2.54% better than CWBA in terms of TCP goodput on the average. In the second simulation topology MCWBA performs 0.49% worse than CWBA again in terms of total TCP goodput achieved by several TCP flows.

Both CWBA and MCWBA algorithms require the availability of the value of *cwnd* at the burst assembler. However, this information is not available in the data packets of the TCP versions considered in this thesis. Therefore, the implementation of our proposals depend on a minor modification that should be done in the TCP header. There are three unused bits in the TCP header and one of them may be used to inform the burst assembler that a packet is the last packet of the sender's window or not for a direct implementation of the CWBA and MCWBA. A second solution would be to estimate the *cwnd* values at the burst assembler. However, this solution may degrade the performance of the proposed algorithms since *cwnd* estimation would make some unavoidable errors.

Another issue to be mentioned about CWBA and MCWBA is the scalability of the algorithms. Both algorithms work on the devices that are capable of per-flow aggregation, which means each TCP traffic flow has its own input buffer at the burst assembler. Due to this configuration, CWBA and MCWBA algorithms are not scalable to the OBS networks with input traffic from hundreds of different TCP flows. The number of flows that the proposed assemblers may handle is limited by the number of available input queues. Especially, CWBA and MCWBA performs much better than the existing burst assemblers in the cases where small number of large size TCP flows inject a high amount of traffic into an OBS network.

The rest of the thesis is organized as follows. Chapter 2 focuses on OBS and the burst assembly algorithms. TCP's congestion control mechanism and the proposed algorithms are presented in Chapter 3. The network models used in the simulations and also the simulation results are given in Chapter 4. Finally, Chapter 5 concludes the thesis.

Chapter 2

OPTICAL BURST SWITCHING

In OBS networks, there are three types of nodes: ingress, egress and core. Ingress/egress nodes are responsible to control the input/output traffic to/from the OBS network. Core nodes manage the traffic between the ingress and egress nodes. Depending on its function, an OBS node may behave as one, two or three of the node types at the same time. The ingress node aggregates the incoming packets from one of its links (can be electrical, optical or wireless link) into optical bursts. The ingress node also sends the bursts together with the corresponding BCPs into the OBS network. Since only one BCP is generated for several user data packets, the control plane overhead is reduced. BCPs are sent in advance to their bursts and also out of band over a dedicated control channel, which consists of one or several wavelengths on a fiber.

First issue to be mentioned is the dedicated control channel. A fiber can support hundreds of wavelengths. In OBS networks most of these wavelengths are used by the data bursts. The rest of the wavelengths are allocated for the

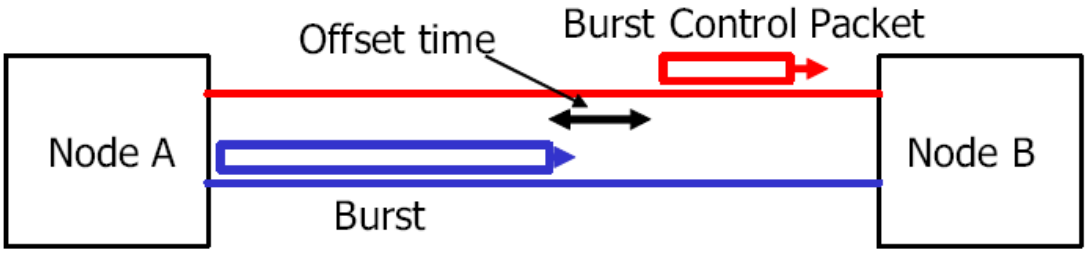

Figure 2.1: Optical burst and control packet

burst control packets. As a result, control and data plane of the OBS networks work independent from each other.

Second issue is the offset time. Offset time is the difference between the sending instant of a BCP and the sending instant of the corresponding burst. The pre-determined offset time basically creates the optical network platform without the need of any optical buffers. To be specific, when a burst is created by the burst assembly mechanism, firstly BCP is generated and sent to the network. BCP packet traverses the optical nodes that the burst will traverse. BCP informs the optical nodes about the coming optical burst. Then depending on the reservation protocol, some of the available bandwidth is reserved for the incoming burst. This scheme is given in Figure 2.1 [3].

The offset time, T_{off}, is a function of the route that is followed by the burst in the OBS network. Let the average processing time per node of the BCP in the OBS core nodes be Δ. For a source destination pair, let the number of hops that will be traversed by the burst be N. Then the offset time, T_{off}, is set to $N\Delta$. By doing so, it is guaranteed that the reservation process is completed at each node before the arrival of the burst. This is because, the time difference between the arrival of BCP and the burst decreases at each hop due to the per hop processing delay. The new offset time is called the residual offset time. For the example given in Figure 2.2 [4], which uses JET that will be explained in the next section, T_{off} is equal to 4Δ at the ingress node. At the first core node, the residual offset time becomes 3Δ, and so on. At the egress node, the residual

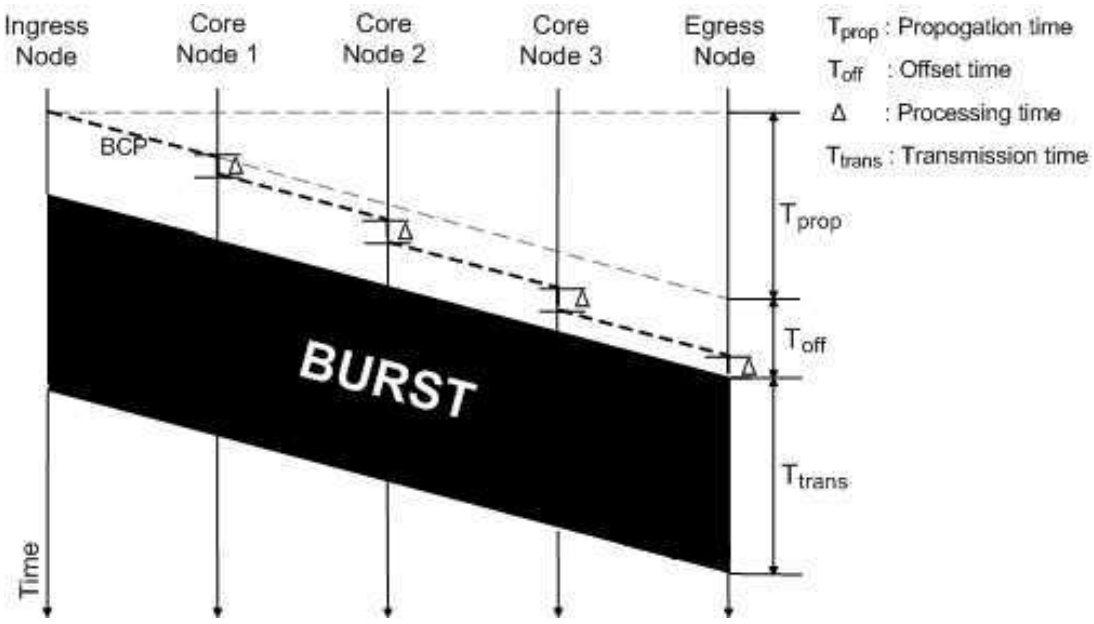

Figure 2.2: Timing diagram of a BCP and the corresponding burst

offset time is equal to zero, which means optical burst arrives at the egress node at the instance of the end of the processing of the BCP. Clearly, adding a guard band in the offset time increases the robustness of the mechanism to timing jitter. Although it is beyond the scope of this thesis, it should be noted that by the proper adjustment of the offset times, quality of service classes can be provided at the ingress nodes.

By the help of the structure explained above OBS networks can handle bursty traffic, and the OBS concept becomes realizable. However, OBS networks suffer from high loss probabilities due to their bufferless nature. When two or more bursts use the same optical link, the concept of burst contention arises. To resolve the contention among the bursts, three solutions are proposed. First contention resolution mechanism is performed in the space domain that is called deflection routing. In this mechanism, different routes are used for the same source-destination pair. Second mechanism is performed in the frequency domain, where contentions are resolved with the wavelength converters. Thus, when there are bursts that use the same link, each burst is assigned to a unique wavelength. The final mechanism is performed in the time domain. The fiber

delay lines (FDLs), which can provide a limited and pre-set delay for the optical data, are used to delay the bursts when there is a contention. Even if all the three contention resolution mechanisms are used in an OBS network, burst losses may occur since the resources are limited in each case.

2.1 Burst Reservation

The reservation protocols for the OBS networks determine the sharing of the available bandwidth between the users and also the duration of the bandwidth assigned to a user. The reservation protocols proposed for OBS networks are Tell-And-Wait (TAW), Tell-And-Go (TAG), Just-In-Time (JIT) and Just-Enough-Time (JET). In TAW protocol, a BCP flows from the source node toward the destination node. If the reservation is successful, an acknowledgment is sent back to the source node. Conversely, a negative acknowledgment is sent to the source node if the reservation fails. The two-way reservation scheme used in TAW do not utilize the available bandwidth efficiently, especially on the links with large propagation delays.

In TAG protocol, data packets and control packets are tightly coupled in time since they both traverse the network simultaneously without an offset time. At the core nodes, the data packets are delayed until the processing of the control packets is finished and the resulting switchings are done. As a result of this structure, optical buffers are needed when TAG protocol is used.

JIT is an open ended reservation protocol. In JIT, there are two types of control packets, namely setup packet and release packet. When a core node receives a setup packet, the desired bandwidth is reserved. This reservation holds until the arrival of the relase packet to the core node.

JET is a close ended one-way reservation protocol. In JET, the burst control packets contain the offset time and the burst length informations. Thus upon the processing of the control packet, the desired bandwidth is reserved from the arrival time of the burst until its departure time. An example of the JET protocol is given in Figure 2.2. Since close-ended reservation achieves the best resource utilization of all, JET based reservation scheme is used throughout this thesis.

2.2 Burst Scheduling

At the core nodes when there is an arrival of a BCP, the scheduling algorithm determines the assignment of the wavelengths to the incoming bursts. The well-known scheduling algorithms in the literature are Horizon and Latest Available Unused Channel with Void Filling (LAUC-VF).

The Horizon algorithm is also referred as Latest Available Unused Channel (LAUC). Horizon of a wavelength is defined as the latest time that the wavelength will be in use. When there is a BCP arrival, LAUC algorithm chooses the wavelength with the minimum horizon that is smaller than the starting time of the burst. The only information about the incoming burst that LAUC needs is the arrival time. When JET reservation protocol is used, the residual offset times of the bursts may be different. Due to these differences and also due to the usage of the FDLs, there may be gaps between the scheduled bursts that are called voids. Minimum horizon approach tries to minimize the voids. However horizon algorithm does not utilize the available bandwidth efficiently because the voids are minimized but they are not used even if the voids are usable by an incoming burst.

The advantage of LAUC-VF algorithm over LAUC is that it also makes use of the generated voids between the scheduled bursts. Therefore in addition to the arrival time of the burst, the duration of the burst is also needed by the void filling

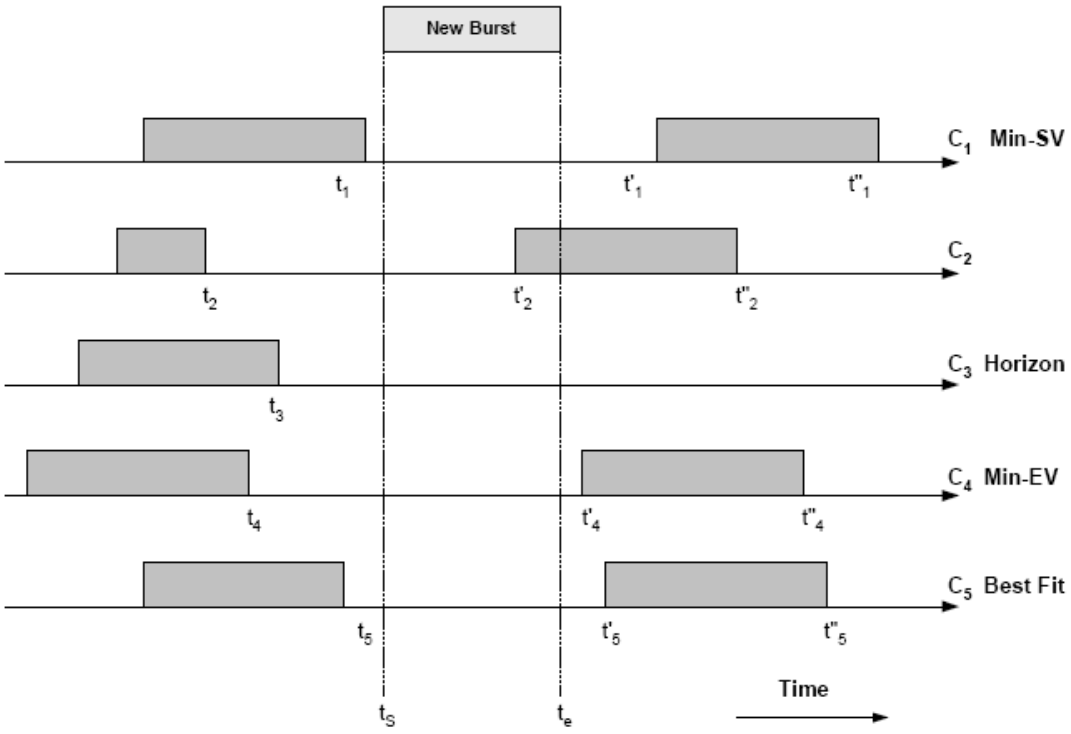

Figure 2.3: Burst scheduling of different algorithms

algorithm. The void filling can be done based on different metrics. Some of the variants in the literature are LAUC-VF with Min-SV (Starting Void), Min-EV (Ending Void) and Best Fit. In LAUC-VF with Min-SV, the algorithm chooses the wavelength among the available ones for which the gap between the end of the scheduled burst and the start of the unscheduled burst is minimum. In LAUC-VF with Min-EV, the available wavelength that has the minimum gap between the end of the unscheduled burst and the start of the scheduled burst is chosen. LAUC-VF with Best Fit tries to minimize the starting and the ending voids that will be introduced after the scheduling of the unscheduled burst [3, 8, 19].

Figure 2.3 depicts the wavelengths that are chosen by different scheduling mechanisms for the arrival of a burst [9]. C_i is the i_{th} wavelength on the output link for $1 \leq i \leq 5$. t_s is the starting time and t_e is the ending time of the unscheduled burst. t_i is the ending time of the first scheduled burst on the i_{th} wavelength, whereas t'_i is the starting and t''_i is the ending time of the second

scheduled burst on the i_{th} wavelength on the output link. If the Horizon algorithm is used, C_3 is chosen since it has the minimum horizon among all the wavelengths. If LAUC-VF with Min-SV is used, C_1 is chosen because $t_1 - t_s$ is the minimum among $t_i - t_s$ for $1 \leq i \leq 5$. Similarly with Min-EV, C_4 is chosen since $t_e - t'_4$ is the minimum among $t_e - t'_i$ for $1 \leq i \leq 5$. When LAUC-VF with Best Fit is used, C_5 is chosen because $t_5 - t_s + t_e - t'_5$ is the minimum of $t_i - t_s + t_e - t'_i$ for $1 \leq i \leq 5$. Finally, C_2 is not chosen in any case due to the fact that the unscheduled burst is in conflict with the already scheduled burst on the 2_{nd} wavelength of the output link.

In the void filling scheduling algorithms the utilization of the resources are better than the horizon algorithms. Moreover, the burst loss probability with the void filling algorithms is less than the one in horizon algorithms. As a result, LAUC-VF with Min-SV scheduling algorithm is used throughout this thesis.

2.3 Burst Assembly

Burst assembly is basically the generation of the optical bursts that will flow in the OBS network. At the ingress nodes, the incoming packets are sorted based on their destinations and priority classes. The arriving packets are put into the appropriate electronic queues of the ingress node. Based on the design of the OBS network, there may be several input queues. For instance, there may be one queue for each of the egress nodes. Moreover, there may be one queue for each priority class of the input traffic. In this case, if k denotes the number of egress nodes and n denotes the number of priority classes, the ingress node contains $k \times n$ input queues as shown in Figure 2.4.

The instant that the bursts are generated is determined based on the metric that the burst assembly algorithm uses. Thus, the interarrival times and also the lengths of the bursts are designated according to the burst assembly criterion.

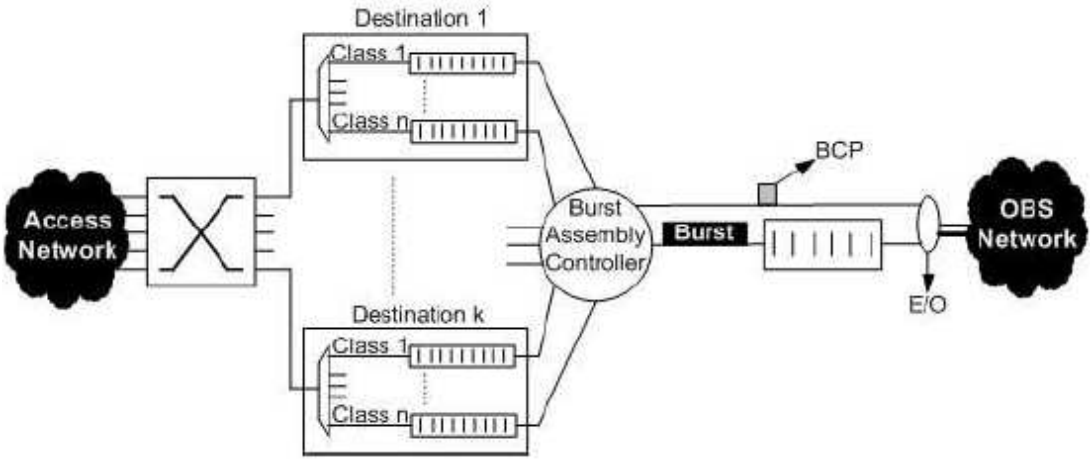

Figure 2.4: OBS ingress node architecture

In another words, the burst assembly mechanism shapes the input traffic based on its decision rule. The burst assembly algorithms proposed in the literature can be classified into four categories based on the metrics they use.

2.3.1 Timer Based Burst Assembly

The well-known timer based assembler starts a timer when a data packet arrives to the ingress node. When the timer hits the pre-set assembly period threshold (timeout) value, all the packets contained in the burstifier queue are aggregated into a burst. First BCP and then after the offset time, the burst is sent to the egress node [20]. Obviously, in the timer based algorithm the interarrival times of the bursts are fixed and the lengths of the bursts are randomly changing. This is because the burst sizes are directly related to the input traffic rate. When the input traffic rate is low, relatively small sized burst are generated. On the other hand when the traffic rate is high, large bursts are generated in the burstifier.

Packet Arrival Event
if *(Corresponding burstification queue timer is not running)*
Start the corresponding timer with the pre-defined timeout value;
end if
Assemble the packet to the corresponding burstification queue;
Update the burst length information;
Timeout Event
Send BCP on a control channel;
Schedule data burst to be sent on a data channel after offset time;
Stop the corresponding burstification queue timer;

When the traffic flowing through the OBS network is generated by TCP, timer-based assembly algorithm is not the optimum algorithm. If the congestion window *cwnd* of TCP is small, timer-based algorithm gives rise to an extra end-to-end delay in the TCP flow. This is because the number of packets coming in the timeout interval of timer-based algorithm is too few and the TCP packets in the burstifier queue waits unnecessarily long. On the other hand, if the *cwnd* size is large, the timeout interval of timer-based algorithm expires before all packets in the current congestion window arrives to the burstifier queue. This results in the generation of more than one bursts for the current window of TCP, which increases the overhead in the network and decreases the correlation gain since small sized bursts are generated.

2.3.2 Burst Length Based Burst Assembly

When there is a packet arrival at the ingress node, the user packet is queued in the related burstifier queue. If the queue has user data at least as much as the pre-defined minimum burst size, the burst is generated [21]. In the length based burst assembly algorithm, the burst lengths are almost constant but the interarrival times of the bursts randomly change. When the input traffic rate is high, the

length based assembler works efficiently. However if the input traffic rate is low, e. g. when TCP congestion window is small, the length based assembler waits too long that the utilization of the OBS network is very low.

Packet Arrival Event

Assemble the packet to the corresponding burstification queue;

Update the burst length information;

if *(Assembled data length* $\geq$ *pre-defined minimum burst length)*

Send BCP on a control channel;

Schedule data burst to be sent on a data channel after offset time;

Reset the burst length;

end if

2.3.3 Mixed Timer/Burst Length Based Burst Assembly

In the mixed assembler, the burst generation event is triggered by the timeout event of the assembly timer for low input traffic rates. Alternatively, it is triggered by the exceeding of the minimum burst length threshold for high input traffic rates [22]. Since this algorithm is a hybrid of the two former algorithms; the interarrivals times of the bursts and also the burst lengths are both randomly changing.

Packet Arrival Event

if *(Corresponding burstification queue timer is not running)*

    Start the corresponding timer with the pre-defined timeout value;

end if

Assemble the packet to the corresponding burstification queue;

Update the burst length information;

if *(Assembled data length $\geq$ pre-defined minimum burst length)*

    Send BCP on a control channel;

    Schedule data burst to be sent on a data channel after offset time;

    Reset the burst length;

    Stop the corresponding burstification queue timer;

end if

Timeout Event

Send BCP on a control channel;

Schedule data burst to be sent on a data channel after offset time;

Reset the burst length;

Stop the corresponding burstification queue timer;

2.3.4 Adaptive Burst Assembly

The burst assembly algorithms defined so far are static algorithms that do not adjust themselves according to the input traffic. The adaptive burst assembly algorithms form the bursts with the triggering of their criterion event, where the criterion parameter is updated with the incoming traffic's properties. For instance, the criterion may be the assembly timeout period and it may be updated according to average burst length, average delay, path loss or any application specific criterion. Thus, in this thesis we focus especially on adaptive burst assembly algorithms since they outperform the static assembly algorithms.

A literature overview about the adaptive burst assembly algorithms is given below. In [11], the burst assembly periods are updated according to the average burst length. The average burst lengths are calculated in a similar fashion to TCP's round trip time calculation. Then, using the most recently calculated average burst length, the new assembly period is computed within a finite interval designated by pre-set variables. [23] and [24] propose the same burst assembly algorithm, where the burst sizes are adaptively increased/decreased if the burstification queue has more/less packets than the pre-set maximum/minimum queue size. In [25], the burst assembly period is selected among three pre-set values according to the congestion window size of TCP. [26] uses the learning automata in the burst assembly. With the help of active measurements in pre-defined discrete time intervals, the probabilities of choosing one value from the assembly periods vector are adjusted. The dynamic threshold assembly algorithm in [27] is based on the average delay of the packets comprising a burst, where a running average delay estimator value is compared with the pre-defined average assembly delay. Improved version of this algorithm is proposed in [28], where four dynamic assembly algorithms are proposed. Among these algorithms, the best algorithm uses traffic prediction to maximize the average length of the bursts produced for a given average burstification delay using a pre-defined lower bound on the length of the bursts. A dynamic algorithm that adapts the burst sizes by changing the assembly periods using the observations of the link losses is proposed in [29]. Similarly, instead of link losses, path losses that are found via active measurements in the network, are used in the assembly algorithm of [30].

Zhou et. al [31] proposed an algorithm that improves the performance of TCP over OBS networks by limiting the ratio of the number of ACK packets to the TCP segments in an optical burst to a fixed value. In [32], the state of the TCP is estimated according to the incoming traffic. Accordingly, the assembly period is updated for the next assembly with the limitation of a maximum pre-defined variable. However, the state estimation is only for a simple implementation

of TCP without *fast recovery* state. Moreover, the authors of [32] performed simulations on a simple network. As a result, the performance of the algorithm is shown to be beneficial for only some of the cases studied in the paper.

All of the adaptive algorithms proposed so far have some pre-defined parameter. Therefore, these algorithms need parameter adjustments based on the topology that they are running. To exterminate the adjustment requirements for different network topologies, we propose a new adaptive assembly algorithm that use TCP's congestion window as the main criterion. Thus our assembly algorithm do not use any pre-set parameters unlike the other burst assembly algorithms in the literature since TCP's congestion window is already an adaptive parameter.

We present our algorithm in the next chapter after a brief TCP summary that is followed by performance analysis of TCP traffic over OBS networks. Then a variant of CWBA algorithm is also given. At the end congestion window estimation techniques that we used in the proposed algorithms are presented.

Chapter 3

CONGESTION WINDOW BASED BURST ASSEMBLY FOR TCP OVER OBS NETWORKS

Most of today's Internet traffic is carried using Transmission Control Protocol (TCP). As a result, TCP traffic flowing through OBS networks has taken great interest in the literature. In this chapter, we first discuss the performance of TCP over OBS networks. We then introduce two variants of an adaptive burstification algorithm for TCP traffic in OBS networks, namely congestion window based burst assembly algorithm. Congestion window estimation technique that we used in our simulations concludes the chapter.

3.1 TCP Overview

TCP is a connection-oriented service provided by the Internet. TCP uses a congestion control algorithm, which adaptively regulates the sending rate of the TCP source. TCP uses end-to-end congestion control mechanism, which do not use any network-assisted congestion control. TCP uses the congestion window (*cwnd*) to impose a constraint on the rate at which a TCP sender can send traffic to the network. To be specific, the amount of unacknowledged data at a TCP sender cannot exceed the minimum of the *cwnd* and the advertised receive window (*awnd*) of the TCP destination. *awnd* is used to prevent the overflow of the TCP receiver buffer. Assuming no overflow of the buffers, the TCP sender's traffic sending rate is roughly equal to *cwnd* packets in Round Trip Times (RTT) seconds [33].

Once the TCP connection is established between the peers, the factors that determine the instantaneous transmission rate of the TCP sender are *cwnd* size and the end-to-end delay. The increase of *cwnd* value depends on the reception of cumulative ACK packets. The decrease of *cwnd* is triggered via the loss of the packets and/or the excessive delays on the path of the packet. The end-to-end delay is affected by a lot of factors such as queuing delay, transmission delay, propagation delay and processing delay. In OBS networks, the burst assembly process is also an important factor since it increases the end-to-end delay between the source-destination pair.

TCP continuously computes sample RTT for the segments that have been transmitted only once. Then based on the sample RTTs, the estimated RTTs are calculated, which is an exponentially weighted moving average. Moreover, the deviation between the estimated and the sample RTTs are found. Hence using the deviation and also the estimated RTT, timeout is calculated, which is called the Retransmission Timeout (RTO) interval. Upon the transmission of a packet

if no acknowledgment is received during the RTO interval, TCP assumes that the packet is lost and retransmits the packet. This event is called an *RTO event*. Another event that TCP assumes the loss of a packet is the *fast retransmission event*. When the TCP sender receives three duplicate ACK packets for the same segment, TCP takes this as an indication of the fast retransmission event. Thus, the relevant segment is retransmitted before the segment's timer expires.

After the connection establishment or after an *RTO event*, TCP is in *Slow Start (SS)* phase and the value of *cwnd* is set to one Maximum Segment Size (MSS). In *SS* state, TCP sender increases its *cwnd* value by one MSS for each acknowledged segment until a segment is lost or $cwnd > ssthresh$, where *ssthresh* is the *Slow Start Threshold*. In other words, *cwnd* is doubled every RTT, which is an exponentially growing rate. When $cwnd > ssthresh$, TCP switches its state to *Congestion Avoidance (CA)* phase. In *CA* phase, TCP sender increases its *cwnd* value by $1/cwnd$ for each acknowledged segment. This means *cwnd* value is increased by one MSS every RTT. Therefore, in *CA* state *cwnd* value linearly grows.

When an *RTO event* occurs, TCP enters the *SS* phase, sets the *ssthresh* value to the half of *cwnd* value and *cwnd* to one MSS. However when three duplicate ACKs are received, which is *Triple Duplicate ACK (TDA) event*, the reaction of TCP depends on the TCP flavor used. In TCP Tahoe, the reaction to RTO events and TDA events are exactly the same. In the newer implementations of TCP that are considered in this thesis, an additional mechanism exists, which is the *fast recovery* mechanism. In TCP Reno, after a *TDA event*, *ssthresh* is set to half of *cwnd* value, *cwnd* value is set to $max(cwnd/2, 1)$ and TCP enters to *CA* phase. TCP NewReno differs from TCP Reno in the case of multiple losses within a window. When multiple packets of a window are lost, TCP Reno halves its *cwnd* value for each *TDA event* and eventually an *RTO event* occurs. However, TCP NewReno retransmits one packet for each ACK that indicates

the next lost packet. Therefore, TCP NewReno does not face with an *RTO event* when multiple packets of a window are lost. TCP SACK uses the same mechanisms with TCP Reno to adapt the *cwnd* value. In addition, TCP SACK uses the option field of the TCP header to indicate the portion of the correctly received sender's window at the receiver by selective acknowledgments. Thus, TCP SACK may retransmit multiple segments at once if necessary.

3.2 Performance of TCP in OBS Networks

Yu et. al [18] has given the performance analysis of TCP traffic over OBS networks for different TCP flavors. The first performance metric of TCP over OBS is *Loss Penalty (LP)*. *LP* is the reduction in the TCP throughput due to the burst losses. Intuitively, TCP throughput decreases as the burst loss rate increases. Let B be the TCP throughput in segments per second, W_m the maximum TCP window size in segments, b the number of ACKed rounds before the sending window is increased, p the burst loss rate in the OBS network and S the number of segments from a TCP flow contained in a burst. *LP* is given by

$$LP\ Ratio \triangleq \frac{B(without\ loss)}{B(with\ a\ burst\ loss\ rate\ p)} \approx W_m\sqrt{\frac{2bp}{3S}} \tag{3.1}$$

for a small loss rate p.

Second metric is the *Delay Penalty (DP)*, which is mainly the delay in time caused by the burst assembly delay. This is because the packets arriving at the ingress nodes face an extra delay until the generation of the burst. Therefore, the TCP *Round Trip Time (RTT)* is increased by the burst assembly and there is a reduction in the throughput. As the burst assembly time increases, TCP throughput decreases. If RTT denotes the round trip time of the TCP flow in seconds and RTT_o the round trip time of the TCP flow without the burst

assembly, the *Delay Penalty* due to burstification is defined as

$$DP\ Ratio \triangleq \frac{B(without\ burst\ assembly)}{B(with\ burst\ assembly)} \approx \frac{RTT}{RTT_o} \tag{3.2}$$

Since TCP guarantees the delivery of the user data; after a burst loss, TCP retransmits the lost segments. In OBS networks, segment losses occur consequently. Therefore, *fast recovery* mechanism of TCP plays an important role on TCP performance. Since TCP SACK may retransmit multiple lost segments in one round, there is no difference between the packet losses and burst losses for TCP SACK. For this reason, TCP SACK does not have any TCP throughput reduction due to the retransmissions. However, TCP Reno and NewReno may have some TCP throughput reduction due to the retransmissions, which is called *Retransmission Penalty (RP)*. In other words, *RP* is the prolonged retransmission period that is caused by multiple retransmission rounds, during which fewer new packets can be sent. For TCP NewReno with a small loss rate p and a large S value, *RP* is given by

$$RP\ Ratio \triangleq \frac{B(one\ retransmission)}{B(S\ retransmissions)} \approx 1 + \sqrt{\frac{3Sp}{2b}} \tag{3.3}$$

For TCP Reno when $\log \sqrt{\frac{2S}{bp}} > S$, *RP* is given by

$$RP\ Ratio \triangleq \frac{B(one\ retransmission)}{B(S\ retransmissions)} \approx \sqrt{3}(1 + \sqrt{\frac{Sp}{2b}}) \tag{3.4}$$

for a small loss rate p. In the second case for TCP Reno, i. e., $\log \sqrt{\frac{2S}{bp}} < S$, *RP* is given by

$$RP\ Ratio \approx \sqrt{3}[1 + (\frac{RTO}{RTT} + \log \sqrt{\frac{2S}{bp}}) \times \sqrt{\frac{p}{2bS}}] \tag{3.5}$$

for a small p, where *RTO* denotes the TCP retransmission timeout value in seconds. It is noted that when $\log \sqrt{\frac{2S}{bp}} > S$, *RP Ratio* for TCP NewReno is always less than the one for TCP Reno, which in turn shows that TCP NewReno exhibits a higher throughput than TCP Reno. Similarly when $\log \sqrt{\frac{2S}{bp}} < S$ and the ratio in (3.3) is less than the ratio in (3.5), TCP NewReno again exhibits a

higher throughput than TCP Reno over OBS networks. In the latter case, i. e., $\log\sqrt{\frac{2S}{bp}} < S$ and $(1+\sqrt{\frac{3Sp}{2b}}) > \sqrt{3}[1+(\frac{RTO}{RTT}+\log\sqrt{\frac{2S}{bp}})\times\sqrt{\frac{p}{2bS}}]$, TCP Reno has a better throughput than TCP NewReno.

The final metric for TCP performance is *Delayed First Loss (DFL) Gain*. Because of the burst assembly process, there is a delay in time before a TCP sender receives an indication of a lost segment. This extra delay gives the TCP sender the chance of increasing its congestion window size to a higher value than the case without burst assembly. As a result, a higher throughput is achieved, which is the *DFL Gain*. *DFL Gain* is given by

$$DFL\ Gain \triangleq \frac{B(first\ loss\ is\ delayed)}{B(first\ loss\ is\ not\ delayed)} \approx \sqrt{S} \tag{3.6}$$

for a fixed small burst loss rate p.

Correlation Gain (CG) is another popular TCP performance metric used in the literature, which is the combination of the *DFL Gain* and *RP*. To sum up, *Loss Penalty*, *Delay Penalty*, *Retransmission Penalty* and *Delayed First Loss Gain* determine the performance of TCP traffic flowing through OBS networks. If T_b denotes the burst assembly time, the optimum burst assembly time, i. e., T_b^{opt} is defined as:

$$T_b^{opt} = \arg\max_{T_b}\{\frac{DFL\ Gain}{RP\times DP}\} \tag{3.7}$$

for a given burst loss rate p [18].

3.3 cwnd Based Burst Assembly (CWBA)

To overcome the deficiencies of the timer-based assembly algorithm for TCP traffic, we propose *cwnd based burst assembly algorithm (CWBA)*. In this assembler, a burst is generated whenever all the packets of a TCP flow's window reaches the ingress node. For instance, if *cwnd* of a TCP flow is equal to 4 packets, optical burst is generated at the instance of the arrival of the last (fourth)

packet of the TCP flow's window to the burstification queue. In this manner, the burst lengths are variable and equal to the length of *cwnd* packets plus the burst header. Moreover, the delay penalty introduced by the assembly process is smaller than the timer-based burst assembly process. This is due to the fact that in CWBA, *cwnd* value increases faster than timer-based assembly for TCP traffic, especially in *slow start (SS)* phase. As a result, after a loss event *cwnd* of the TCP flow increases much faster than timer-based assembler.

Packet Arrival Event
Assemble the packet to the corresponding burstification queue;
Update the burst length information;
if *(Number of packets in the assembled data* $\geq$ *cwnd size (packets))*
    Send BCP on a control channel;
    Schedule data burst to be sent on a data channel after offset time;
    Reset the burst length;
end if

Another benefit of the CWBA algorithm is that its performance is independent of the TCP flavor. In timer-based burst assembly with TCP Reno, NewReno or SACK, when a burst is lost, with a probability greater than zero the loss event triggers the TCP to enter the *fast recovery* state. Thus the flavor of TCP plays a role in the performance of TCP traffic in OBS network. In other words, the differences between these three TCP implementations come from the *fast recovery* mechanisms of TCP and their interactions with the burst assembly mechanism in OBS networks. TCP SACK has the best performance in OBS networks with timer-based burst assembly. TCP NewReno performs better than TCP Reno if the burst sizes are relatively small and vice versa [18].

In the CWBA algorithm, when a burst is lost, the whole window of TCP flow is lost. Thus with probability one, TCP *retransmission timeout* event occurs and TCP never enters the fast recovery state. As a result, the flavor of TCP does

not have an impact on the performance of TCP traffic in OBS networks with CWBA.

CWBA algorithm requires that the last packet of each TCP sender's window is known to the burst assembler. To implement this algorithm, one can use one of the unused bits in the TCP header. This bit is set to inform the ingress node that the last packet of the current *cwnd* is sent. Whenever the burst assembler receives a TCP packet with the corresponding bit set, a burst is generated. Otherwise, the assembler waits for the reception of the incoming packets from the TCP sender's window. In this manner, TCP header must be accessed by CWBA, which means CWBA needs high access speed to packet headers including transport layer packets.

In addition, for each TCP flow injecting traffic into OBS network, a new session of the CWBA algorithm is needed. This is because each flow's traffic are aggregated into their own assembly queue waiting for the generation instant of the burst. Therefore, CWBA algorithm runs on per-flow aggregation schemes. Thus CWBA may handle several TCP flows limited by the number of assembly queues at ingress node. For instance, CWBA algorithm may not be implemented at the ingress nodes that have hundreds of HTTP traffic from different flows. On the other hand, CWBA becomes the proper choice where an ingress node carries TCP traffic from a few flows with large *cwnd* values. Grid applications, FTP servers with high upload and/or download traffic in the order of hundreds of Mbits/s are the potential applications where CWBA increases TCP goodput significantly.

3.4 Mixed cwnd/Timer Based Burst Assembly (MCWBA)

In CWBA, the burst sizes are variable. In fact, the burst sizes are directly related to *cwnd* size of the TCP sender. Although CWBA algorithm significantly improves the goodput compared with the timer based burst assembly, the burst loss probability increases as the burst size gets larger. In our case, when the *cwnd* value of the TCP sender is a large value, the corresponding burst sizes become very large, which face high burst drop probabilities.

Furthermore, the burst size gets larger as *cwnd* gets higher, which in turn increases the round-trip time for the TCP flow. With CWBA, the variance in the round-trip time of the TCP sender is larger than the case with the timer based algorithm. This condition has a negative impact on the TCP performance. A maximum assembly period threshold can be used by the burst assembler. As a result, we propose to use a combination of timer based assembly and CWBA algorithms, which is called *mixed cwnd/timer based burst assembly (MCWBA)* algorithm.

In the MCWBA algorithm, the bursts are generated whenever the assembly period expires or all packets in the congestion window are buffered in the ingress node, whichever is earlier.

Packet Arrival Event

if *(Corresponding burstification queue timer is not running)*

Start the corresponding timer with the pre-defined timeout value;

end if

Assemble the packet to the corresponding burstification queue;

Update the burst length information;

if *(Number of packets in the assembled data* $\geq$ *cwnd size (packets))*

Send BCP on a control channel;

Schedule data burst to be sent on a data channel after offset time;

Reset the burst length;

Stop the corresponding burstification queue timer;

end if

Timeout Event

Send BCP on a control channel;

Schedule data burst to be sent on a data channel after offset time;

Reset the burst length;

Stop the corresponding burstification queue timer;

3.5 cwnd Estimation

Both CWBA and MCWBA algorithms use the *cwnd* size of the TCP sender. For this reason, the knowledge of the congestion window at the ingress nodes become indispensable for the algorithms that are proposed in this thesis. If the proposals are to be used without any modification in TCP, the estimate of *cwnd* should be calculated and used at the ingress nodes of the OBS networks. To achieve this goal, congestion window estimation techniques in the literature are investigated [34]-[42]. Basically there are two approaches to estimate *cwnd*. In the first approach, firstly an RTT estimate (RTT_{est}) is found at the estimation point. Then the number of packets that come during the RTT_{est} gives the *cwnd*

estimate ($cwnd_{est}$). There are several techniques to calculate RTT_{est} value. To find a proper RTT_{est} value, some extra time is required but the extra time delays the time instant that $cwnd_{est}$ is available and therefore the first approach is not suitable for our purposes. In the second approach, the state of TCP is replicated at the estimation point. To replicate the state, sequence numbers and ACK sequence numbers are needed. Thus in the second approach, TCP header must be accessible by the algorithm that will estimate the *cwnd* values. However, it should be noted that the second approach cannot be implemented if secure TCP connections are used, e. g., SSL.

Among these *cwnd* estimation techniques, [40, 41, 42] is chosen since the algorithm is clearly explained and the error rates are relatively small. In this estimator, a replica of the state of the TCP sender's state is constructed in the measurement point in the form of a finite state machine. In this manner, $cwnd_{est}$ is found. Moreover, at the measurement point the TCP flavor is found among TCP Tahoe, Reno and NewReno. The estimate is found from the Receiver-to-Sender ACK packets. In this technique, underestimation and also overestimation of *cwnd* is possible like all the other *cwnd* estimation algorithms. In the application of the algorithm, the authors see 5-15% error in the *cwnd* estimates. In addition with the help of *cwnd* estimate, it is found that which ACK packet triggers the new packet to be sent. Thus, RTT estimates are also calculated. For the RTT_{est} value, the estimation error never exceeds 25%.

In CWBA, the algorithm presented below is used for the *cwnd* estimation of TCP Reno. It is possible to implement the *cwnd* estimation algorithms for the other implementations of TCP, as well. However, current implementation of the estimator is not suitable to work online in a burst assembler. This is because whenever there is a packet loss between the packet sender and the measurement point, $cwnd_{est}$ may be under/overestimated as a consequence of where the packet is lost. Therefore to see the performance of the proposed algorithms with $cwnd_{est}$,

we assume a network topology with no TCP packet loss between TCP sender and measurement and also no ACK packet loss between measurement point and TCP sender. Thus we are able to run the *cwnd* estimator as an online estimator.

```
Initialize cwnd, ssthresh, state, awnd, dupACKnum
ACK Packet Arrival Event
if (Default state)
    if (New packet)
        if (cwnd < ssthresh)
            cwnd++
        else
            cwnd+=1/cwnd
        end if
    else if (Duplicate ACK)
        dupACKnum++
        if (dupACKnum ≥ 3)
            ssthresh=max(min(awnd,cwnd)/2,2)
            cwnd=cwnd/2 + 3
            state=FastRecovery
        end if
    end if
else if (FastRecovery state)
    if (New packet)
        cwnd=ssthresh
        dupACKnum=0
        state=Default
    else if (Duplicate ACK)
        dupACKnum++
        cwnd++
    end if
end if
```

```
TCP Packet Arrival Event
if (Default state)
    if (RTO packet)
        ssthresh=max(min(awnd,cwnd)/2,2)
        cwnd=1
    end if
else if (FastRecovery state)
    if (RTO packet)
        ssthresh=max(min(awnd,cwnd)/2,2)
        cwnd=1
        dupACKnum=0
        state=Default
    end if
end if
```

The algorithms presented so far namely CWBA, MCWBA and CWBA with $cwnd_{est}$ are compared with each other and also with the timer based burst assembler via the simulations performed in nOBS in the following chapter. The simulation results are explained thoroughly and also the superiority of the proposed algorithms are shown in the upcoming chapter.

Chapter 4

SIMULATION RESULTS

In order to see the effects of burst assembly on TCP performance, we first use a simple OBS network topology shown in Fig. 4.1. OBS switches use only a single

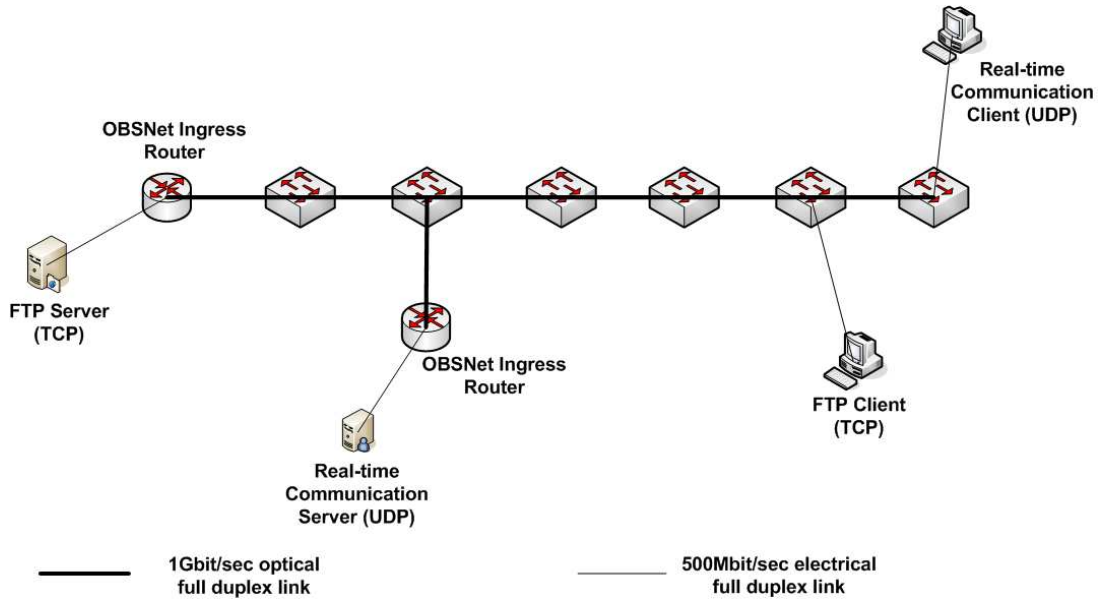

Figure 4.1: Simple OBS network topology

wavelength for data bursts and there are no fiber delay lines. The maximum number of packets that can be aggregated into a burst is set to 10000. Each ingress node of the OBS network has one burstification queue with a queue length of 100000 packets (per-flow aggregation). The FTP server employs an

infinite FTP flow, which sends TCP segments through OBS network to the FTP client. The MSS of TCP source is set to 1040 bytes and the receive windows are set to 10000 MSS. ACK packets triggered by TCP flow do not experience any drop or assembly delay and packet size of ACKs are 40 bytes.

Real-time communication server injects exponentially distributed traffic into the network, which is carried in UDP segments. On the average 1% of the time, real-time communication server sends UDP packets with a rate of 500 Mbps into the network. Therefore, the only reason for burst losses in the OBS network is due to the contention between the TCP carrying bursts and UDP carrying bursts. Furthermore, the OBS ingress node connected to the real-time communication server uses timer based burst assembly algorithm with an assembly period of 4 msec. Optical links have 1 Gbps bandwidth and 1 msec propagation delay. Electrical links have 500 Mbps bandwidth and 1 msec propagation delay. Optical burst switches have 5 μsec switching time and 200 μsec per-hop processing delay.

We also use a second OBS network topology, which is the mesh network shown in Fig. 4.2. In this topology, all the variables are the same as in the first topology except that the access links of the ingress (N9-N11) and egress (N12-N14) nodes have 0.1 msec propagation delay whereas the optical links between the OBS core nodes have 2.5 msec propagation delays. Each FTP server S_i employs an infinite FTP flow to FTP client D_i for $1 \leq i \leq 9$. In this network, burst losses occur because of the contention between nine optical burst flows.

We use nOBS, which is an ns2 based OBS network simulation tool [15]. JET reservation protocol is used in conjunction with Latest Available Unused Channel with Void Filling (LAUC-VF) as the scheduling algorithm. We performed simulations for the network models in Fig. 4.1 and Fig. 4.2 with timer based burst assembly algorithm for different assembly periods, CWBA and MCWBA algorithms. When timer based or MCWBA algorithm is used, all the ingress nodes use the same assembly timeout. Also to see the impact of the flavor of

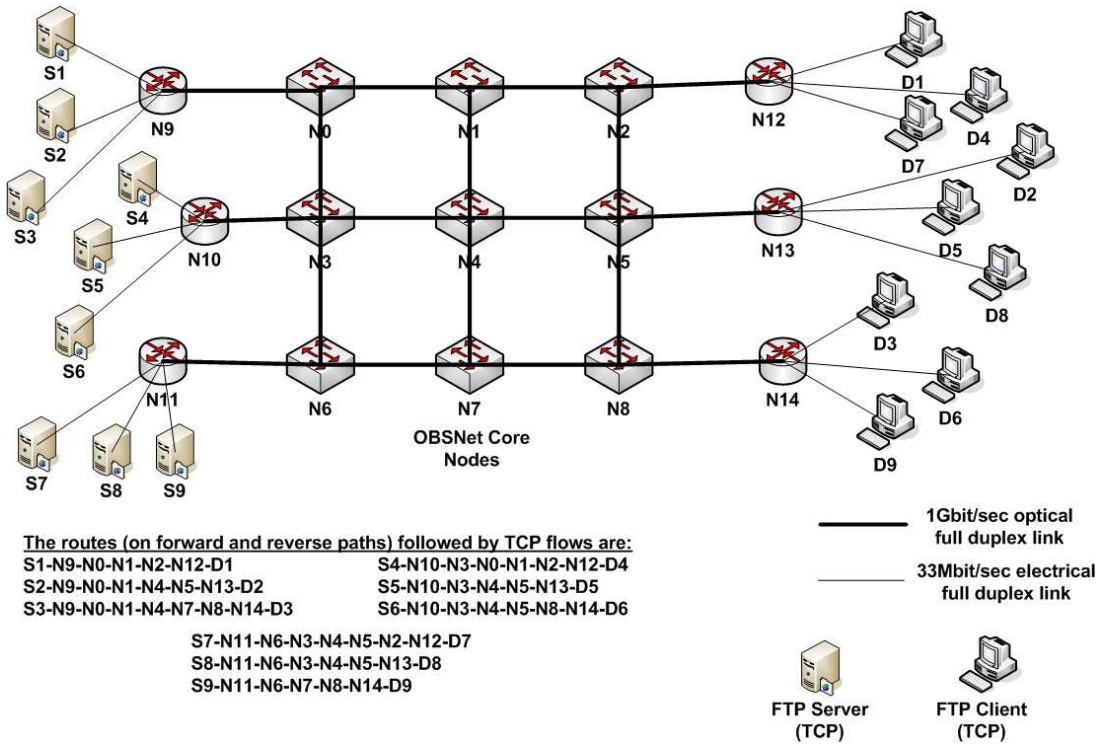

Figure 4.2: Mesh OBS network topology

TCP on TCP performance, TCP Reno, TCP NewReno and TCP SACK are used. In the first part of the simulations, we assume that the exact value of the TCP congestion window is known by the burst assembler. We later relax this requirement by using a *cwnd* estimation algorithm at the assembler that utilizes passive measurements.

4.1 CWBA with Simple Topology

As mentioned before, CWBA algorithm is independent of the flavor of TCP and therefore it performs exactly the same for TCP Reno, NewReno and SACK. The TCP goodputs achieved by the timer based and CWBA algorithms for the simple OBS network topology are compared in Fig. 4.3. As the assembly timeout of the timer-based assembly algorithm increases, TCP goodput first increases due to the correlation gain and then decreases due to excessive assembly delay. This behavior is observed for all three TCP versions tested. On the other

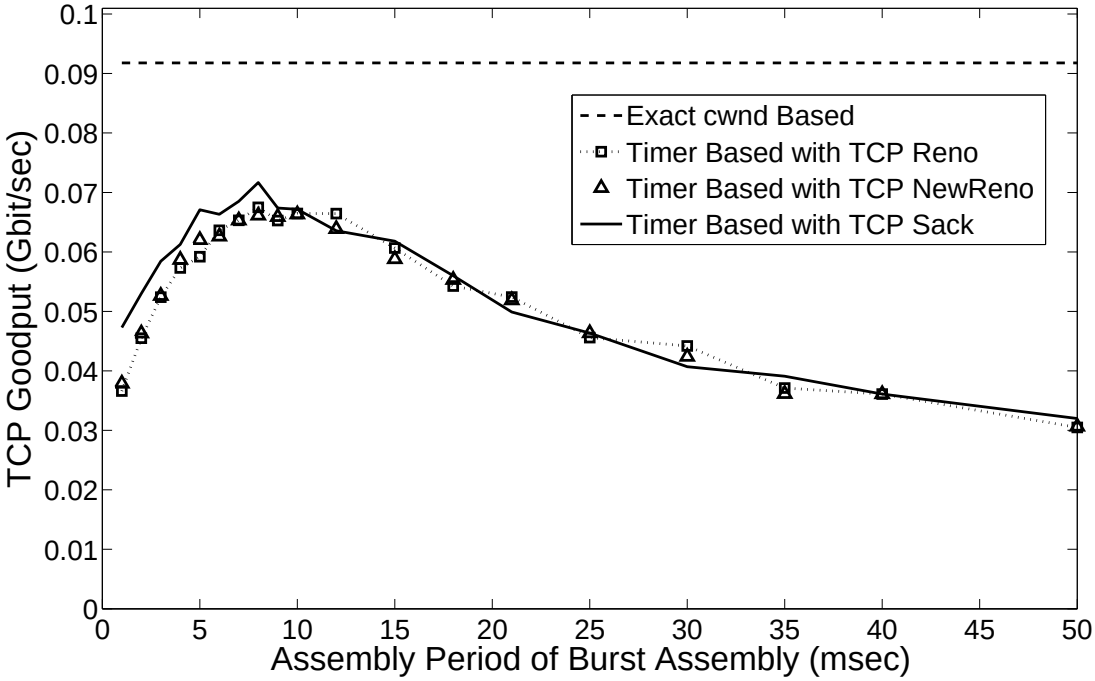

Figure 4.3: Performances of timer based and CWBA algorithms for the simple OBS network topology

hand, CWBA algorithm achieves about 28% improvement in TCP goodput when compared with the case of using the timer-based burst assembly together with the optimum value of the timeout and TCP SACK. The performance improvement percentages of CWBA algorithm with respect to timer based algorithm with optimum assembly periods are given in Table 4.1.

Table 4.1: Goodput improvement of CWBA algorithm for the simple OBS network topology

Reference Algorithm	Goodput Improvement
Timer Based Burst Assembly with TCP Reno	36.01%
Timer Based Burst Assembly with TCP NewReno	38.4%
Timer Based Burst Assembly with TCP SACK	28.09%

Samples of congestion window evolutions are given in Fig. 4.4 and Fig. 4.5 for CWBA and timer based burst assembly algorithms, respectively. The values of *cwnd* for the timer based assembly shown in Fig. 4.5 are obtained using TCP Reno and the optimum assembly delay of 8 msec. TCP's *cwnd* increases approximately 50% faster with CWBA compared with the timer based assembly in the *slow start*

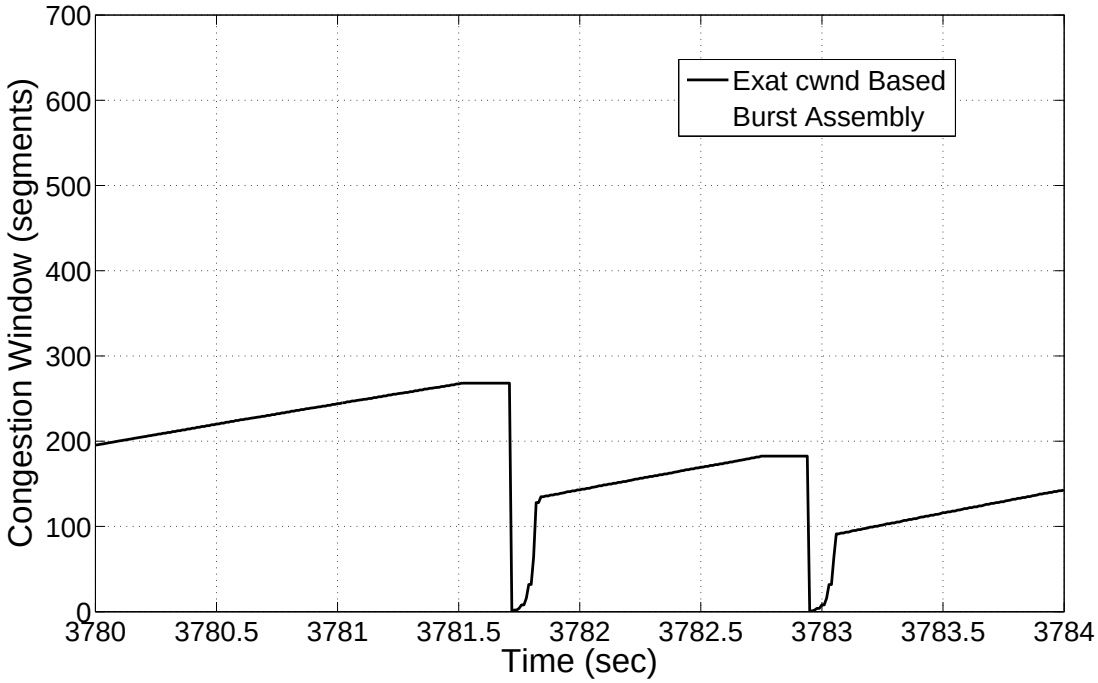

Figure 4.4: Congestion window evolution of CWBA algorithm

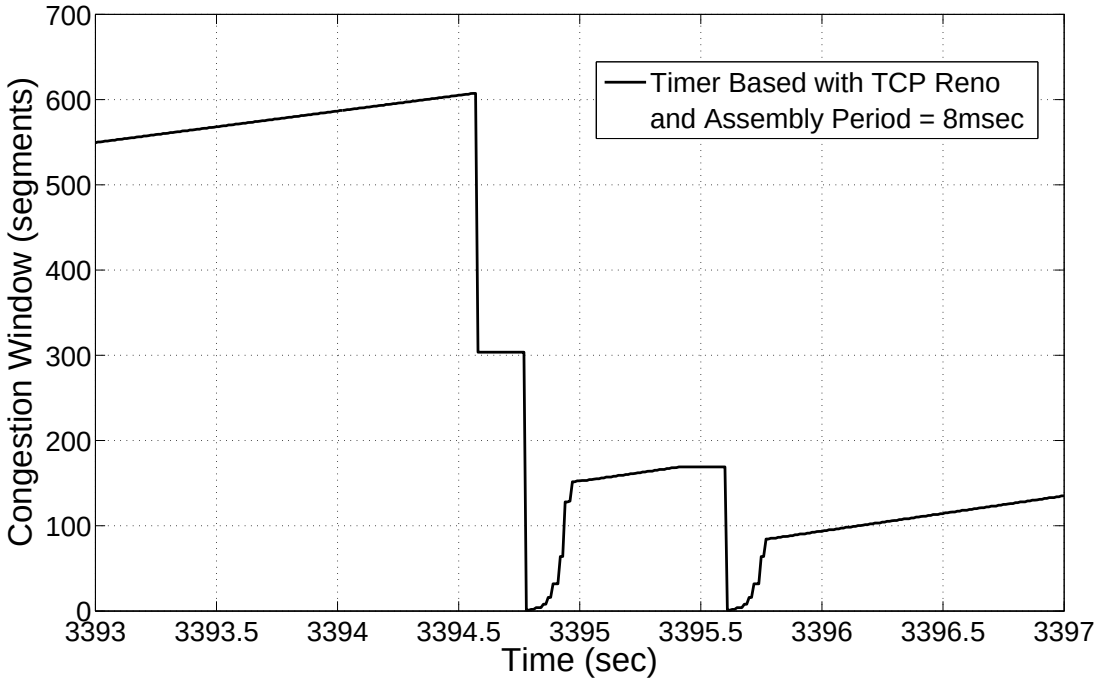

Figure 4.5: Congestion window evolution of timer based assembly algorithm

phase. In addition to this, the timer based assembly algorithm experiences TCP's fast retransmission and fast recovery states, whereas with CWBA only timeout loss events occur. It is also observed that in *congestion avoidance* phase both algorithms have similar slopes.

The implementation of CWBA algorithm requires some modifications at the TCP layer since some fields in the TCP headers of incoming packets needs to be marked accordingly. In order to remove this limitation, we propose to use an estimate of the *cwnd* value at the burst assembler. To this end, we use the *cwnd* estimation algorithm of Jaiswal et. al [40, 41, 42], which creates a replica of the state of the TCP sender's state at the measurement point using passive measurements that is given in Section 3.5. Although this algorithm is not an online algorithm, it is suitable for our case since no loss model on the links are used in the simulations.

As far as OBS networks are considered, the closest node to the TCP sender is chosen to execute this algorithm, which is the ingress node corresponding to the TCP sender node. The simulations are repeated with using the *cwnd* estimation algorithm at the burst assembler instead of making the exact value of the *cwnd* available at the assembler. The simulations show that there is a 1.02% reduction on goodput when CWBA uses the estimated *cwnd* value compared with CWBA which uses the exact *cwnd* value. This degradation is due to the errors in the *cwnd* estimation.

4.2 CWBA with Mesh Topology

In addition to the simple OBS network topology, we also performed simulations with a mesh OBS network topology. The reported results are obtained by repeating each simulation three times. In this topology there are 9 TCP flows destined to the TCP clients. The routes used by TCP flows are shown in Fig. 4.2. In order

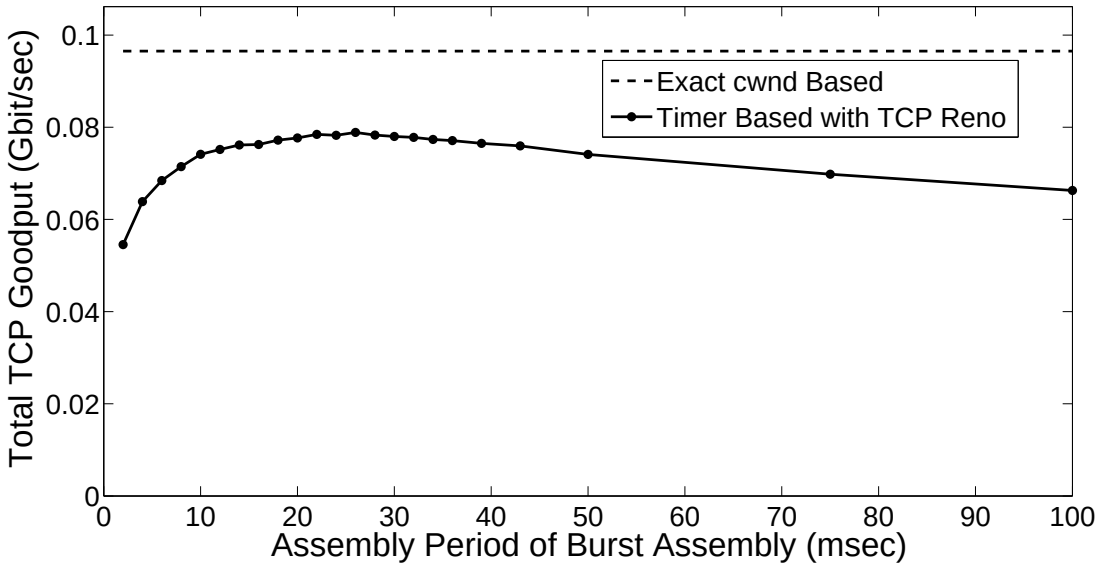

Figure 4.6: TCP performance of timer based and CWBA algorithms in the mesh OBS network topology

to eliminate the possibility of synchronization between TCP flows when timer based assembly is used, the assembly period is added with a zero-mean normally distributed random variable with a standard deviation of 10% of the corresponding average assembly period. The average of the total goodput achieved by these TCP flows for both assembly algorithms are given in Fig. 4.6. In this scenario,

Table 4.2: Performance gain of CWBA to timer based assembly algorithm with the optimum assembly period in the mesh OBS network topology for each TCP flow

	Perf. Gain	AP_{opt}	Contribution
Flow 1	12.02%	26 msec	21.05%
Flow 2	17.87%	18 msec	6.98%
Flow 3	21.62%	36 msec	10.99%
Flow 4	11.43%	20 msec	5.48%
Flow 5	34.3%	32 msec	13.77%
Flow 6	13.23%	10 msec	7.91%
Flow 7	15.63%	16 msec	6.62%
Flow 8	6.67%	50 msec	9.68%
Flow 9	19.22%	30 msec	17.52%

CWBA algorithm performs 22.36% better than the timer based algorithm on the average. The performance gain for each TCP flow, which is given by the improvement of TCP goodput with CWBA algorithm compared with the timer

based algorithm that uses the optimum assembly period (AP_{opt}) are given in Table 4.2. AP_{opt} values for timer based algorithm and also the contribution of each flow on the total goodput for CWBA algorithm are also reported.

The burst loss probabilities and the average burst lengths for the same scenario are given in Fig. 4.7 and Fig. 4.8, respectively. The shapes of the curves for TCP goodput and burst loss probabilities show a great resemblance. For the timer based assembler, as the assembly period increases average burst length increases as well. Similarly, the burst loss probability increases with the increasing assembly period as the offered traffic to the network increases. However, as the assembly period increases further, the loss probability decreases as the offered load from TCP flows decreases due to increasing assembly delay and there is less contention between bursts. As a conclusion, the simulation results show that the performance gain of CWBA algorithm is not because of the enhancement of the burst loss probability but it is mainly due to the improvement in the assembly delay.

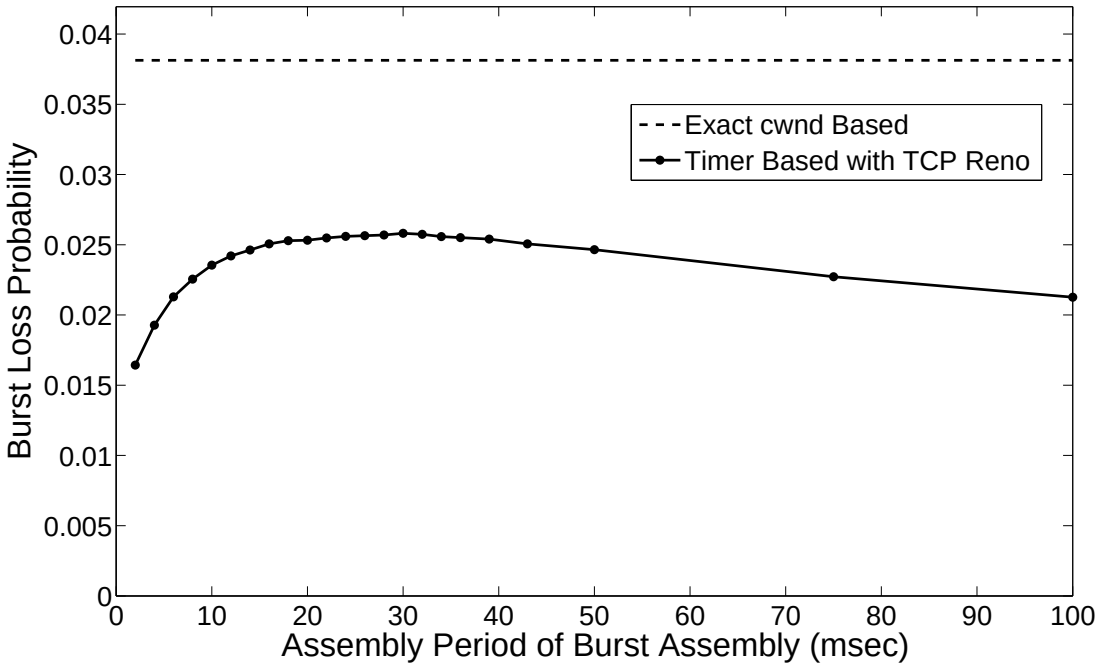

Figure 4.7: Burst loss probabilities for timer based and CWBA algorithms in the mesh OBS network topology

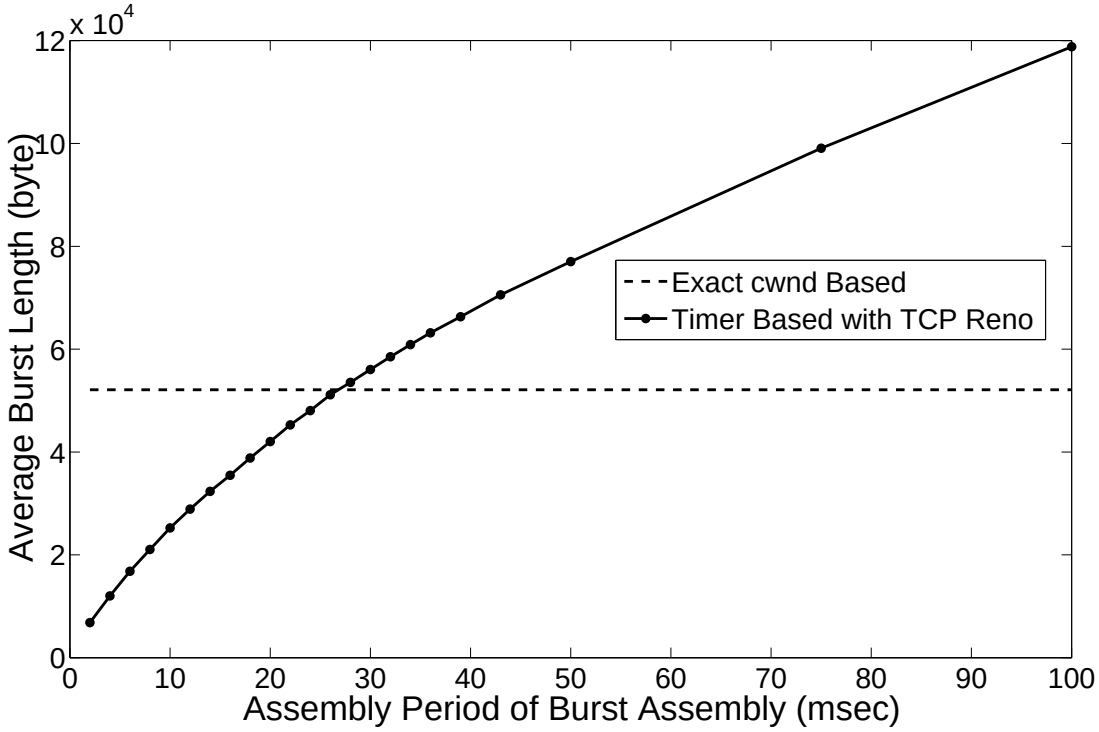

Figure 4.8: Average burst lengths for timer based and CWBA algorithms in the mesh OBS network topology

4.3 MCWBA with Simple Topology

MCWBA simulations are performed using the topology in Fig. 4.1. Each simulation is repeated five times with different random variable sets. At the end, the averages of these runs for CWBA and MCWBA algorithms are given in Fig. 4.9. In this case, mixed burst assembly algorithm enhances the goodput by 2.54% with respect to CWBA.

TCP goodputs corresponding to two different TCP flows are shown in Fig. 4.10 and Fig. 4.11, corresponding to an optimistic and a pessimistic case, respectively. In the optimistic case, the usage of timer in CWBA is almost always beneficial. On the other hand, in the pessimistic case the usage of timer in CWBA algorithm does not provide any advantage.

After a certain value of the assembly period value, MCWBA and CWBA behaves exactly the same. This is explained as follows: let the transmission time

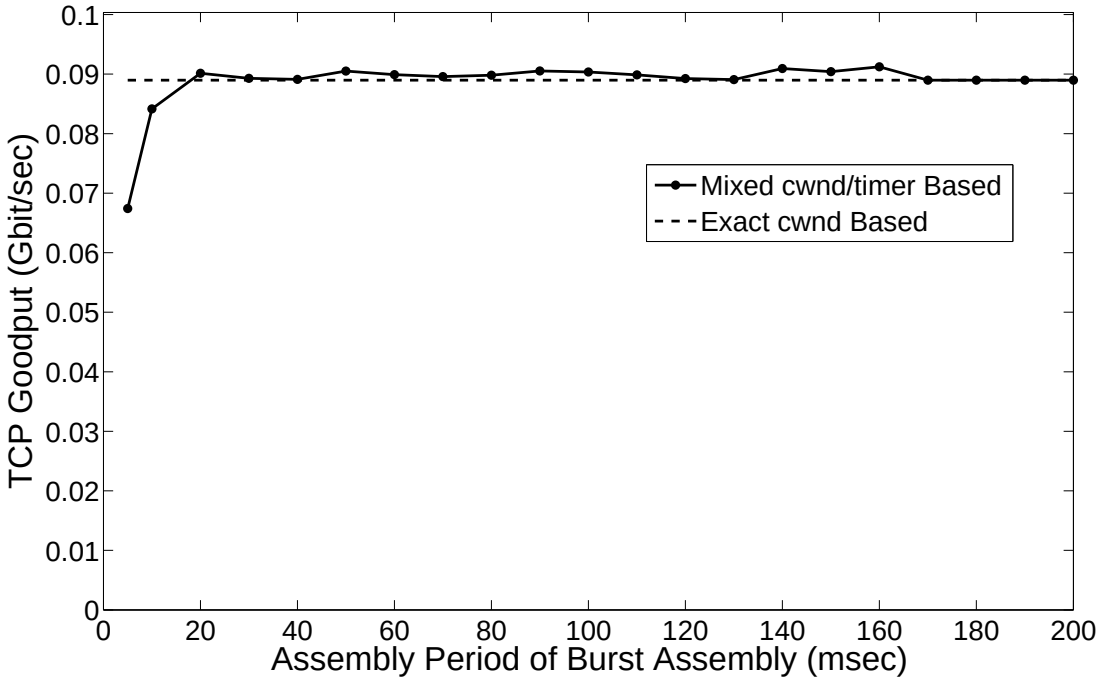

Figure 4.9: TCP performance of CWBA and MCWBA algorithms (average of 5 simulations) in the simple OBS network topology

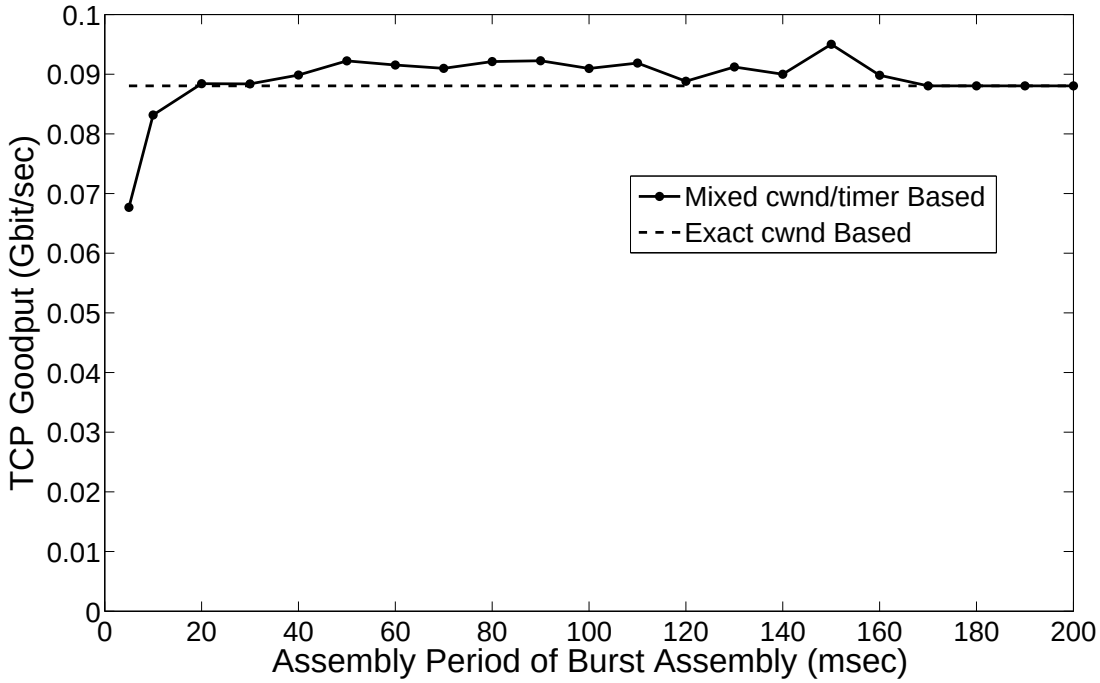

Figure 4.10: TCP performance of CWBA and MCWBA algorithms (optimistic case) in the simple OBS network topology

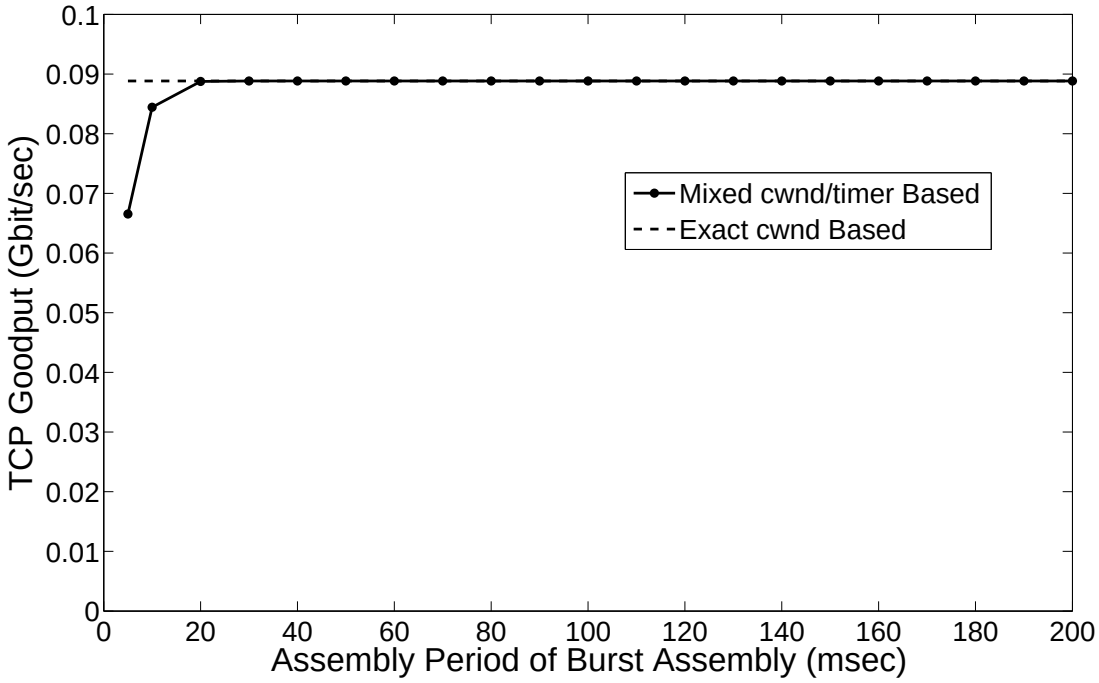

Figure 4.11: TCP performance of CWBA and MCWBA algorithms (pessimistic case) in the simple OBS network topology

of the largest burst be T_{trans}, switching time of OBS switches be T_{sw} and the per-hop processing delay be T_{hprc}. In this manner, when the assembly period is larger than $T_{trans} + T_{sw} + T_{hprc} = T_{max}$, MCWBA algorithm converges to CWBA algorithm. In our simulations with the simple topology, T_{max} can be calculated as

$$T_{trans} = \frac{\min(\max(cwnd), \max(rec.window))(PacketSize)}{AccessLinkBandwidth}$$

$$T_{trans} = \frac{(10000 packets)(1040 \frac{bytes}{packet})(8 \frac{bits}{byte})}{500 Mbps}$$

$$T_{trans} = 166.4 \; msec$$

resulting in

$$T_{max} = 166.4 \; msec + 5 \; \mu sec + 200 \; \mu sec = 166.605 \; msec$$

As a result, when the assembly period exceeds 166.605 msec, MCWBA generates exactly the same goodput as CWBA. We conclude that with a proper choice of the assembly period, MCWBA algorithm performs much better than

the existing assembly algorithms and it is also more robust than CWBA algorithm in the sense that it has an upper bound on the delay introduced by the burst assembly process.

4.4 MCWBA with Mesh Topology

MCWBA and CWBA algorithms are also compared using the mesh topology given in Fig. 4.2. The total goodput achieved by the TCP flows for both assembly algorithms can be seen in Fig. 4.12. Although, MCWBA algorithm guarantees a maximum assembly delay for the TCP flows, the total goodput achieved by nine TCP flows is 0.49% worse with MCWBA algorithm compared with CWBA algorithm. However, MCWBA performs better than CWBA for some of the TCP flows. As an example, the goodput for the fourth TCP flow is shown in Fig. 4.13. The performance improvement of MCWBA obtained using the optimum values of the timer value (AP_{opt}) with respect to CWBA and the TCP goodput contribution of each flow to total TCP goodput of all flows are reported in Table 4.3

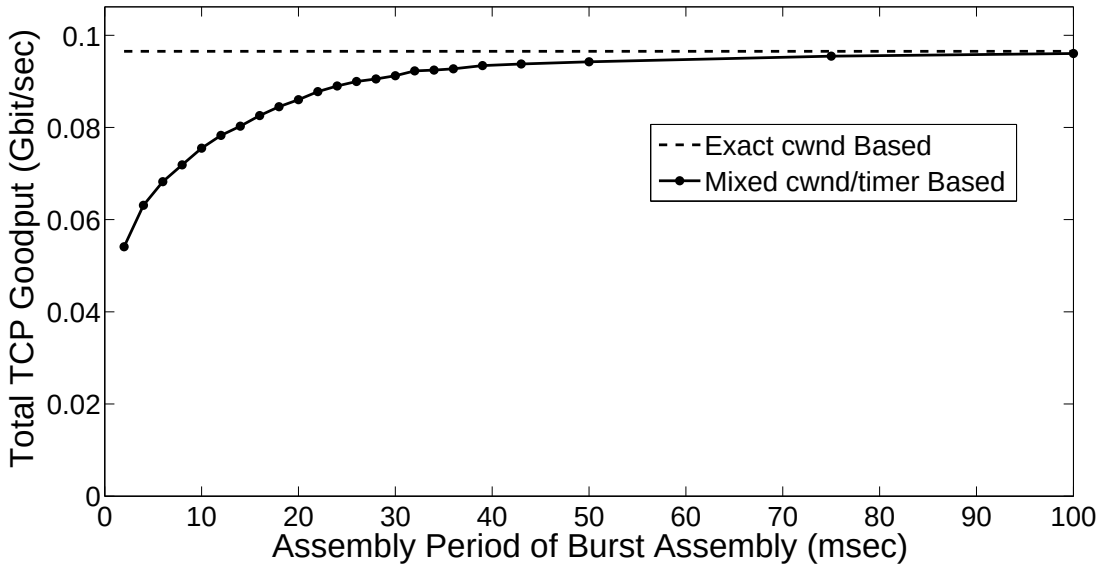

Figure 4.12: TCP performance of CWBA and MCWBA algorithms for the mesh OBS network topology

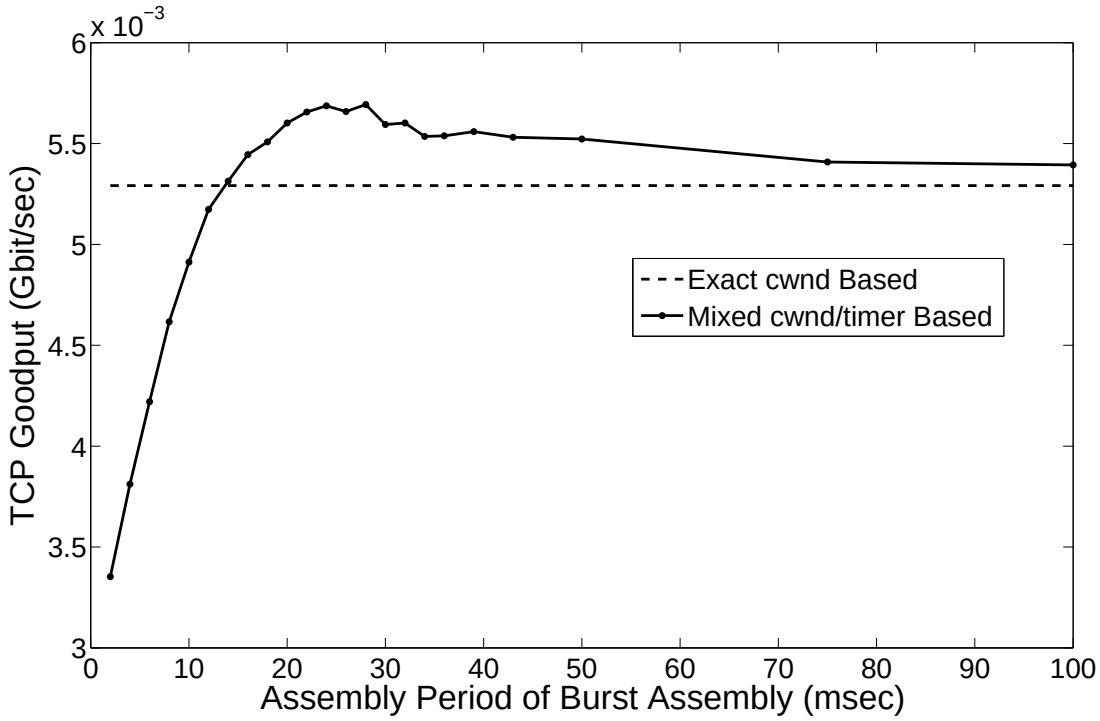

Figure 4.13: TCP performance of CWBA and MCWBA algorithms for the 4^{th} TCP flow for the mesh OBS network topology

Table 4.3: Performance gain of MCWBA compared with CWBA for each TCP flow in the mesh OBS network topology

	Perf. Gain	AP_{opt}	Contribution
Flow 1	1.6%	100 msec	21.51%
Flow 2	-4.03%	75 msec	6.72%
Flow 3	1.47%	75 msec	11.24%
Flow 4	7.91%	28 msec	6.3%
Flow 5	-0.52%	100 msec	13.68%
Flow 6	3.66%	28 msec	8.63%
Flow 7	0.41%	43 msec	6.83%
Flow 8	2.52%	100 msec	9.98%
Flow 9	0.24%	100 msec	17.64%

The burst loss probabilities of the fourth TCP flow and average loss probability of all flows for the mesh topology are depicted in Fig. 4.14. The loss probability for the fourth flow is higher than the average loss probability. This shows that since the loss probability of the fourth flow is higher, its congestion window takes smaller values on the average than the other flows. Therefore, for the fourth flow small assembly periods yields a better performance. Thus for the fourth TCP flow with MCWBA algorithm, smaller bursts, which face a relatively low burst loss rate, are generated than the case with CWBA algorithm. As a consequence, MCWBA performs better than CWBA for the fourth flow. As a future work, the effect of the usage of different assembly algorithms for different TCP flows may be investigated.

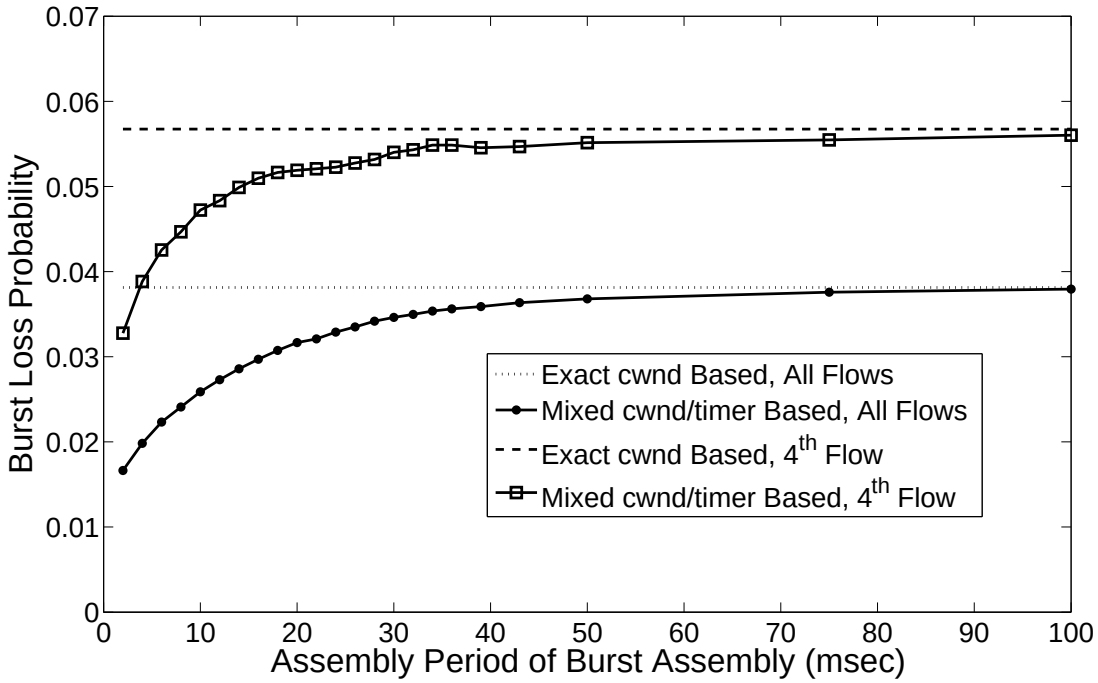

Figure 4.14: Burst loss probabilities for CWBA and MCWBA algorithms in the mesh OBS network topology

In addition to the burst loss probabilities, the average burst lengths of all TCP flows for the mesh topology is shown in Fig. 4.15. For MCWBA algorithm, as the assembly period increases, the average burst length increases and eventually converges to the average burst length of CWBA algorithm. This explains

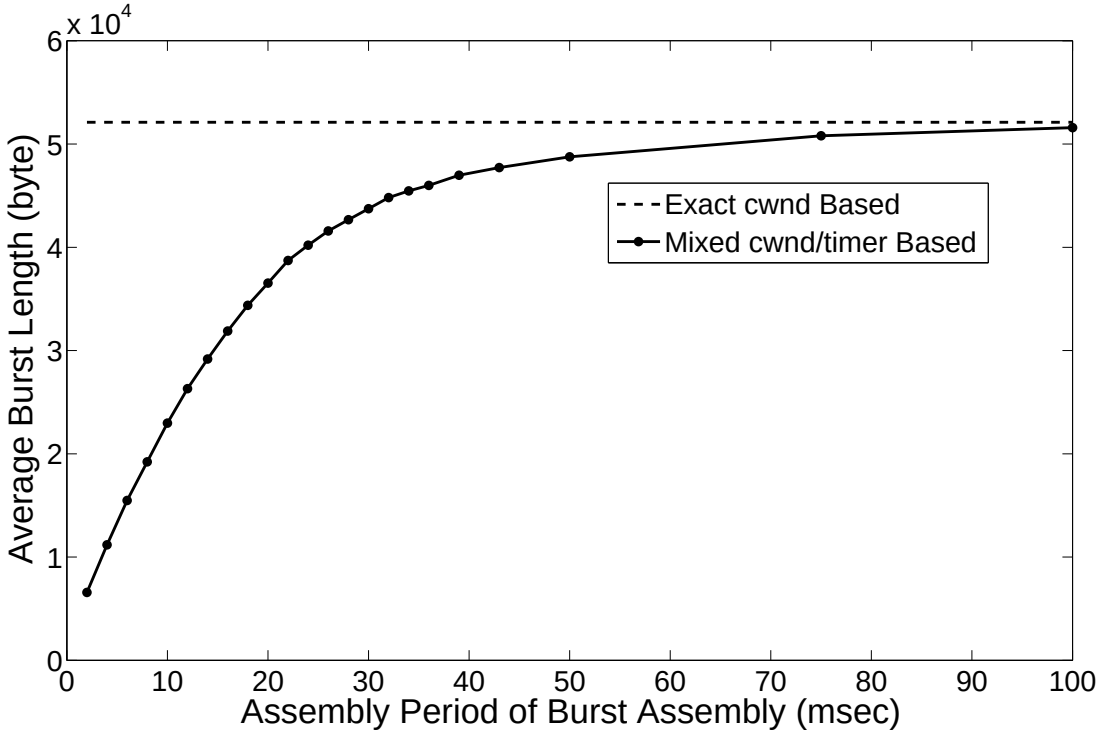

Figure 4.15: Average burst lengths for CWBA and MCWBA algorithms in the mesh OBS network topology

why the burst loss probability of the mixed algorithm converges to the loss probability of CWBA algorithm as the assembly period increases. This also shows that in MCWBA algorithm, the average burst length never exceeds the average burst length of CWBA algorithm. Therefore, after the loss events, the *cwnd* part of the mixed algorithm prevents the average *cwnd* values of the flows to decrease too much, which is not the case in the timer based algorithm. As a result, as the assembly period increases the burst loss probability of MCWBA algorithm converges to the loss probability of CWBA algorithm in a monotonically increasing fashion.

Chapter 5

CONCLUSIONS

Optical burst switching is one of the most promising candidates to be deployed in the optical networks in the near future. An important element that affects the performance of the OBS networks is the burst assembly algorithm. In this thesis, we proposed two new adaptive optical burst assembly algorithms for TCP traffic in OBS networks: *cwnd* based burst assembly (CWBA) and mixed *cwnd/timer* based burst assembly (MCWBA) algorithms. Both algorithms significantly improve the TCP goodput with respect to the timer based burst assembly algorithms. The performance gain achieved by CWBA with respect to timer based burst assembler goes up to approximately 40%. MCWBA algorithm performs better than CWBA algorithm on the average about 2%. However, the robustness and the performance improvement of MCWBA comes with the increased complexity.

Both of the proposed assembly algorithms need the knowledge of *cwnd* size of the TCP sender, which can be implemented by using one of the unused bits in the TCP header. Alternatively by using *cwnd* estimation techniques, the algorithms can be implemented at the cost of increased complexity and a slight performance degradation, but without the requirement of any modification in

the TCP header. However, in two of the cases TCP header is investigated by CWBA or MCWBA. This results in the necessity of high access speeds to the transport layer packets. Moreover, CWBA and MCWBA are designed to work on the burst assemblers with per-flow aggregation. As a result both algorithms are favorable in the situations with a few number of TCP flows with a large amount of traffic.

The adaptive burst assembly algorithms in the literature have different performance metrics than TCP goodput. Therefore, a comparison between the adaptive assemblers in the literature and CWBA/MCWBA have not been made. As a future work, the adaptive assemblers in the literature may be implemented and a thorough performance comparison may be done. In addition, *mixed cwnd/burst length based burst assembly* algorithm may be defined and investigated in terms of TCP goodput performance using the same simulation topologies given in this thesis. Combinations of CWBA and other assembly algorithms may also be investigated. Finally, *cwnd* estimation techniques may be expanded to work for different flavors of TCP other than TCP Reno. The estimation algorithm should also be improved in such a way that it is an online algorithm and it works with any kind of link losses in the network.

Bibliography

[1] F. Keçeli, *"Wavelength Assignment in Optical Burst Switching Networks Using Neuro-Dynamic Programming."* M. S thesis, Bilkent University, September 2003.

[2] K. Doğan, *"Performance Study of Asynchronous/ Synchronous Optical Burst/ Packet Switching with Partial Wavelength Conversion."* M. S thesis, Bilkent University, February 2006.

[3] A. K. Kamçı, *"Effect of Burst Length on Loss Probability in OBS Networks with Void-Filling Scheduling."* M. S thesis, Bilkent University, September 2006.

[4] O. Öztürk, *"Quality of Service Analysis for Slotted Optical Burst Switching Networks."* M. S thesis, Bilkent University, January 2008.

[5] T. Battestilli and H. Perros, "An Introduction to Optical Burst Switching," *IEEE Optical Communications*, August 2003.

[6] L. Xu, H. Perros and G. Rouskas, "Techniques for Optical Packet Switching and Optical Burst Switching," *IEEE Communications Magazine*, vol. 39(1), pp. 136-142, January 2001.

[7] J. Li, C. Qiao, J. Xu and D. Xu, "Maximizing Throughput for Optical Burst Switching Networks," *Proc. IEEE INFOCOM'04*, vol. 3, pp. 1853-1863, Hong Kong, China, March 2004.

[8] M. Yoo and C. Qiao, "Optical Burst Switching (OBS)- A New Paradigm for an Optical Internet," *Journal of High-Speed Networks*, vol. 8, no. 1, pp. 69-84, 1999.

[9] Y. Chen, C. Qiao and X. Yu, "Optical Burst Switching: A New Area in Optical Networking Research," *IEEE-Network*, vol. 18, pp. 16-23, May-June 2004.

[10] T. Battestilli and H. Perros, "Optical Burst Switching: A Survey," Tech. rep. TR-2002-10, NC State Univ., Comp. Sci. Dept., July 2002.

[11] X. Cao, J. Li, Y. Chen and C. Qiao, "Assembling TCP/IP Packets in Optical Burst Switched Networks," *Proc. IEEE GLOBECOM'02*, pp. 2808-2812, November 2002.

[12] X. Cao, J. Li, X. Cao, Y. Chen and C. Qiao, "Traffic Statistics and Performance Evaluation in Optical Burst Switched Networks," *IEEE/OSA Journal of Lightwave Technology*, vol. 22(12), pp. 2722-2738, December 2004.

[13] J. Y. Choi, H. L. Vu, C. W. Cameron, M. Zukerman and M. Kang, "The Effect of Burst Assembly on Performance of Optical Burst Switched Networks," *Lecture Notes in Computer Science, Proc. IN*, vol. 3090, pp. 729-739, 2004.

[14] S. Gowda, R. K. Shenai, K. M. Sivalingam and H. C. Cankaya, "Performance Evaluation of TCP over Optical Burst-Switched (OBS) WDM Networks," *Proc. IEEE ICC'03*, pp. 1433-1437, May 2003.

[15] G. Gurel, O. Alparslan and E. Karasan, "nOBS: an ns2 based simulation tool for TCP performance evaluation in OBS Networks," *Proc. Simulation Tools for Research and Education in Optical Networks*, Oct. 2005, France.

[16] *"Network Simulator 2,"* developed by L. Berkeley Network Laboratory and University of California Berkeley, http://www.isi.edu/nsnam/ns

[17] G. Gurel and E. Karasan, "Effect of Number of Burst Assemblers on TCP Performance in Optical Burst Switching Networks," *IEEE BROADNETS'06*, pp. 1-7, October 2006.

[18] X. Yu, C. Qiao, Y. Liu and D. Towsley, "Performance Evaluation of TCP Implementations in OBS Networks," Tech. Rep. 2003-13, CSE Dept., SUNY, Buffalo, NY, 2003.

[19] H. Boyraz, *"Flow Control and Service Differentiation in Optical Burst Switching Networks."* M. S thesis, Bilkent University, April 2005.

[20] A. Ge, F. Callegati and L. S. Tamil, "On Optical Burst Switching and Self-Similar Traffic," *Proc. Opticomm*, vol. 4874, pp. 125-136, 2002.

[21] V. M. Vokkarane, K. Haridoss and J. P. Jue, "Threshold-Based Burst Assembly Policies for QoS Support in Optical Burst-Switched Networks," *IEEE Communications Letters*, vol. 4, no. 3, pp. 98-100, March 2000.

[22] G. Gürel, *"Effect of Burst Assembly over TCP Performance in Optical Burst Switching Networks."* M. S thesis, Bilkent University, July 2006.

[23] S. Oh and M. Kang, "A Burst Assembly Algorithm in Optical Burst Switching Networks," *Proc. Optical Fiber Communication Conf.*, pp. 771-773, 2002.

[24] S. Oh, H. H. Hong and M. Kang, "A Data Burst Assembly Algorithm in Optical Burst Switching Networks," *ETRI Journal*, vol. 24, no. 4, pp. 311-322, August 2002.

[25] K. Ramantas, K. Vlachos, Ó. González de Dios and C. Raffaelli, "TCP Traffic Analysis for Timer-based Burstifiers in OBS Networks," *Lecture Notes in Computer Science, Proc. ONDM*, vol. 4534, pp. 176-185, 2007.

[26] T. Venkatesh, T. L. Sujatha and C. S. R. Murthy, "A Novel Burst Assembly Algorithm for Optical Burst Switched Networks Based on Learning

Automata," *Lecture Notes in Computer Science, Proc. ONDM*, vol. 4534, pp. 368-377, 2007.

[27] K. Christodoulopoulos, E. Varvarigos and K. Vlachos, "A New Burst Assembly Scheme Based on the Average Packet Delay and its Performance for TCP Traffic," *Elsevier Optical Switching and Networking*, vol. 4, iss. 3-4, pp. 200-212, November 2007.

[28] A. Sideri and E. A. Varvarigos, "New Assembly Techniques for Optical Burst Switched Networks Based on Traffic Prediction," *Lecture Notes in Computer Science, Proc. ONDM*, vol. 4534, pp. 358-367, 2007.

[29] B. Kantarci and S. Oktug, "Adaptive Threshold Based Burst Assembly in OBS Networks," *Proc. Canadian Conference on Electrical and Computer Engineering (CCECE 2006)*, pp. 485-488, Ottawa, Canada, May 2006.

[30] B. Kantarci and S. Oktug, "Path Loss Rate Driven Burst Assembly in OBS Networks," *Lecture Notes in Computer Science, Proc. ISCIS*, vol. 4263, pp. 483-492, 2006.

[31] J. Zhou, J. Wu and J. Lin, "Improvement of TCP Performance over Optical Burst Switching Networks," *Lecture Notes in Computer Science, Proc. ONDM*, vol. 4534, pp. 194-200, July 2007.

[32] S. Peng, Z. Li, X. Wu and A. Xu, "TCP Window Based Dynamic Assembly Period in Optical Burst Switching Network," *Proc. IEEE ICC'07*, pp. 2365-2370, June 2007.

[33] J. F. Kurose and K. W. Ross, *"Computer Networking: A Top-Down Approach Featuring the Internet."* Addison-Wesley, 2005.

[34] J. Wu and H. El-Ocla, "TCP Congestion Avoidance Model with Congestive Loss," *Proc. IEEE ICON'04*, vol. 1, pp. 3-8, November 2004.

[35] I. T. Ming-Chit, D. Jinsong and W. Wang, "Improving TCP Performance over Asymmetric Networks," *ACM Comput. Commun. Rev.*, vol. 30, no. 3, pp. 45-54, 2000.

[36] S. Biaz and D. Yawen, "Enhancing Congestion Control for Wireless Links," *Communication Networks and Distributed Systems*, pp. 93-99, January 2003.

[37] H. Liao, Q. Zhang, W. Zhu and Y. -Q. Zhang, "A Robust TCP/IP Header Compression Scheme for Wireless Networks," *Proc. IEEE International Conference on 3G Wireless and Beyond*, San Francisco, CA, June 2001.

[38] R. Lance and I. Frommer, "Round-Trip Time Inference Via Passive Monitoring," *ACM SIGMETRICS Performance Evaluation Review*, vol. 33, no. 3, pp. 32-38, December 2005.

[39] M. Dryna, *"Network Tomography Tools."* M. S thesis, Technische Universität München Fakultät für Informatik and Institut Eurécom Sophia-Antipolis, September 2005.

[40] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose and D. Towsley, "Inferring TCP Connection Characteristics Through Passive Measurements," *Proc. IEEE INFOCOM*, March 2004.

[41] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose and D. Towsley, "Inferring TCP Connection Characteristics Through Passive Measurements (extended version)," Tech. rep. RR03-ATL-070121, Sprint ATL, July 2003.

[42] S. Jaiswal, *""Measurements-In-The-Middle": Inferring End-End Path Properties and Characteristics of TCP Connections Through Passive Measurements."* Ph. D thesis, Graduate School of the University of Massachusetts Amherst, September 2005.