Photon Netw Commun (2010) 20:138–150
DOI 10.1007/s11107-010-0254-2

Congestion window-based adaptive burst assembly for TCP traffic in OBS networks

Seckin Ozsarac · Ezhan Karasan

Received: 16 June 2009 / Accepted: 12 June 2010 / Published online: 29 June 2010

Abstract Burst assembly is one of the key factors affecting the TCP performance in optical burst switching (OBS) networks. When the TCP congestion window is small, the fixed-delay burst assembler waits unnecessarily long, which increases the end-to-end delay and thus decreases the TCP goodput. On the other hand, when the 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. In this paper, we propose adaptive burst assembly algorithms that use the congestion window sizes of TCP flows. Using simulations, we show that the usage of the congestion window size in the burst assembly algorithm significantly improves the TCP goodput (by up to 38.4% on the average and by up to 173.89% for individual flows) compared with the timer-based assembly, even when the timer-based assembler uses the optimum assembly period. It is shown through simulations that even when estimated values of the congestion window size, that are obtained via passive measurements, are used, TCP goodput improvements are still close to the results obtained by using exact values of the congestion window.

Keywords Optical burst switching · Adaptive burst assembly · TCP over OBS

S. Ozsarac
TUBITAK-UEKAE/ILTAREN, 06800 Ankara, Turkey
e-mail: seckin.ozsarac@iltaren.tubitak.gov.tr

E. Karasan (✉)
Department of Electrical and Electronics Engineering,
Bilkent University, 06800 Ankara, Turkey
e-mail: ezhan@ee.bilkent.edu.tr

1 Introduction

Due to recent advances in optical transmission technologies, there is a growing mismatch between extremely high optical transmission rates and relatively slower switching and processing speeds of optical switches. Optical burst switching (OBS) [1–3] is a sub-wavelength transfer mode which is between optical circuit switching and optical packet switching. OBS separates the data and control planes in the optical and electrical domain, respectively, to eliminate the technological problems involved in processing the packet header. In an OBS network, an ingress OBS node assembles data packets emanating from its clients into bursts. A variable-length optical burst is composed of several packets, avoid small size optical packets, so that the stringent requirements for transmission and synchronization in the optical domain can be avoided. Once the optical burst is formed, a control packet which includes the header information is sent, so that the OBS switches along the path from the ingress node to the egress node can be configured beforehand for the incoming burst.

Our focus in this paper is mainly on the burst assembly algorithms for TCP traffic flows. When TCP flows over OBS networks are considered, TCP performance is significantly affected by the burst assembly algorithm. The burst assembly mechanism introduces a delay penalty in TCP throughput since the round-trip times of the TCP flows are increased [4]. This is simply because the incoming packets to the ingress node need to wait in the burstifier queue until the optical burst is generated. On the other hand, the enlargement of the transmission units from single packets to bursts and thus increasing the number of TCP segments between consecutive loss events increase the TCP performance, which is the so-called *correlation gain* [3–7].

In the literature, there are various burst assembly algorithms that can be classified into four categories: timer-based,

burst length-based, mixed timer/burst length-based (hybrid) and adaptive [5,8–16] burst assembly algorithms. The key parameters in the burst assembly algorithm are the maximum and minimum burst sizes and the assembly timer threshold value [17].

The first three categories of burst assembly algorithms are static algorithms that do not adjust themselves according to the input traffic. The adaptive burst assembly algorithms use a set of parameters that are dynamically updated. For instance, the assembly timeout period can be updated based on the average burst length, average delay, path loss or any application specific criterion. In this paper, we focus especially on adaptive burst assembly algorithms since they outperform the static assembly algorithms [5,8–16].

Adaptive burst assembly algorithms are classified into two sub-categories: burst assembly algorithms for a general traffic model [8–12] and burst assembly algorithms for a TCP traffic model [5,13–16]. Among the algorithms for general traffic models, in [8], the burst sizes are adaptively increased/decreased if the burstification queue has more/less packets than the pre-set maximum/minimum queue size. Therefore, pre-defined queue size parameters directly affect the performance of the assembler. An adaptive algorithm that adapts the burst sizes by changing the assembly periods using the observations of the link loss probabilities is proposed in [9]. In this algorithm, the burst sizes can take values from three different intervals that are determined by the pre-defined link loss rate threshold values. A similar algorithm is reported in [10] where path loss probabilities that are calculated using active measurements in the network are used instead of link loss rates. In [11], four adaptive assembly algorithms are proposed. Among these algorithms, the best performing algorithm uses traffic predictions 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. Learning automata are used in [12] for 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, which is finite in size, are adjusted.

The second class of adaptive burst assembly assumes an offered traffic based on TCP. In [5], 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. The adaptive assembly algorithm in [13] 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. In [14], the burst assembly period is selected among three pre-set values according to the congestion window size of TCP. Zhou et al. [15] 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 pre-defined fixed value. In [16], the state of the TCP is estimated with 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 the *fast recovery* state. Moreover, the simulations are performed on a simple network and the algorithm improves the performance for only some of the cases studied in the paper.

All of the adaptive algorithms proposed so far have at least one pre-defined fixed parameter. Therefore, these algorithms need parameter adjustments based on the topology that they are running. We propose a new adaptive assembly algorithm which uses TCP's congestion window as the main criterion, which exterminates the adjustment requirements on different network topologies. The proposed algorithm 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 as it only adapts to the congestion control algorithm of TCP flow. CWBA has the capability to work on any network topology without the need of any adjustment for optimality. The delay penalty introduced by the burst assembly process is significantly reduced with CWBA since the delay penalty due to burst assembly 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 performance of the CWBA algorithm is evaluated through simulations performed in nOBS [18], which is an ns2 [19] based simulation tool for OBS networks. Gurel et al. [20] has shown that the fixed 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. The simulation 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% on average and by up to 173.89% for an individual flow even when the fixed timer-based burstifier uses the optimum value for the burstification timeout. It is also shown that CWBA achieves the same goodput independent of the TCP version when TCP Reno, New-Reno and SACK algorithms are used.

We also propose a hybrid version of the CWBA algorithm and the timer-based burst assembly algorithm, called the mixed *cwnd*/timer-based burst assembly (MCWBA) algorithm. MCWBA is more appropriate for delay sensitive applications since it puts an upper bound on the

delay introduced by the burst assembly process. It is shown through simulations that MCWBA exhibits similar performance with CWBA in terms of total TCP goodput.

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 that 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 carrying a very large number of TCP flows. Instead these algorithms are more appropriate for applications with very large TCP flows, e.g. grid application for e-science.

The rest of the paper is organized as follows. Section 2 introduces the proposed algorithms. The results of the simulations are given in Section 3. Finally, Section 4 concludes the paper.

2 Congestion window-based burst assembly for TCP over OBS networks

When the traffic flowing into the OBS network is generated by TCP, fixed timer-based assembly algorithm is not the optimum algorithm. If the *cwnd* of TCP is small, timer-based algorithm may significantly increase the end-to-end delay for 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 a single window of TCP, which increases the overhead in the network and decreases the correlation gain of burst assembly since small-sized bursts are generated.

2.1 Congestion window-based burst assembly algorithm

To overcome the deficiencies of the timer-based assembly algorithm for TCP traffic, we propose CWBA. In this assembler, a burst is generated whenever all the packets of a TCP flow's window reach 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 size of the congestion window 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 the slow start (SS) phase. As a result, after a loss event *cwnd* of the TCP flow increases much faster than timer-based assembler.

The burst assembly procedure for CWBA is given below:

Packet Arrival Event
Assemble the packet to the corresponding burstification queue
Update the burst length information
if (Number of packets in assembler $\geq$ *cwnd*) **then**
  Send burst control packet (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 [21]. 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 [21].

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 of the unused bits in the TCP header can be used such that the last packet of the current *cwnd* is marked. Whenever the CWBA 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 addition, for each TCP flow injecting traffic into OBS network, a new instance of the CWBA algorithm is needed. This is because each flow's traffic is aggregated into its own assembly queue waiting for the generation instant of the burst. Therefore, CWBA algorithm runs on per-flow aggregation schemes. Per-flow burstification achieves higher TCP goodputs by eliminating synchronization between the congestion control algorithms of TCP flows that share a common burstifier [20]. CWBA may not be suitable for ingress nodes that handle a very large number of small TCP flows (mice) since per-flow burstification requires high implementation complexity when too many flows exist at an ingress node. CWBA is more suitable when an ingress node carries a small number of large TCP flows (elephants). 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 can be applied.

2.2 Mixed congestion window/timer-based burst assembly algorithm

In CWBA, burst sizes are variable as they 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 when a void-filling burst scheduling algorithm such as LAUC-VF is used [20]. In our case, when the *cwnd* value of the TCP sender is large, the corresponding burst sizes may become excessively large, that may increase 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 duration threshold can be used by the burst assembler in order to mitigate these problems. As a result, we propose to use a combination of timer-based assembly and CWBA algorithms, which is called 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.

The burstificaiton procedure for MCWBA is given below:

Packet Arrival Event
if (Corresponding burstification queue timer is not running) **then**
  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 assembler $\geq$ *cwnd*) **then**
  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 Congestion window estimation

Both CWBA and MCWBA algorithms make use of the *cwnd* size of the TCP sender for burstification, which makes the knowledge of the congestion window at the ingress nodes indispensable for these algorithms. If these algorithms 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. There are several congestion window estimation techniques in the literature [22–27], that can be categorized into two groups. In the first approach, an RTT estimate ($\mathrm{RTT_{est}}$) is found at the estimation point. Then the number of packets that come during the $\mathrm{RTT_{est}}$ gives the *cwnd* estimate ($cwnd_{\mathrm{est}}$). There are several techniques to calculate $\mathrm{RTT_{est}}$. To find a proper $\mathrm{RTT_{est}}$ value, some extra time is required. The extra time delays the time instant that $cwnd_{\mathrm{est}}$ 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. 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, [27] 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, $cwnd_{\mathrm{est}}$ 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 overestimation of *cwnd* is possible similar to all the other *cwnd* estimation algorithms. In the application of the algorithm, the authors report 5–15% error in the *cwnd* estimates.

The version of CWBA which uses the estimated *cwnd* values is called estimated *cwnd*-based burst assembly algorithm (ECWBA). The following algorithm is used for *cwnd* estimation of TCP Reno. It is possible to implement the *cwnd* estimation algorithms for other versions of TCP, as well.

Initialize cwnd, ssthresh, state, awnd, dupACKnum
ACK Packet Arrival Event
if (Default state) **then**
  if (New packet) **then**
    if (cwnd < ssthresh) **then**
      cwnd++
    else
      cwnd+=1/cwnd

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

3 Simulation results

We use nOBS, which is an ns2-based OBS network simulation tool [18], for the comparative evaluation of the proposed adaptive assembly algorithms. In the simulations, JET reservation protocol is used in conjunction with Latest Available Unused Channel with Void Filling (LAUC-VF) as the scheduling algorithm. We performed simulations using the timer-based burst assembly algorithm, CWBA and MCWBA algorithms. When timer-based or MCWBA algorithm is used, all the ingress nodes use the same assembly timeout. In order to see the impact of the flavor of TCP on 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 the *cwnd* estimation algorithm at the assembler that utilizes passive measurements.

Two different topologies are used in the simulations. We first use a linear OBS network topology shown in Fig. 1. OBS switches use only a single 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 10,000. Each ingress node of the OBS network has one burstification queue with a queue length of 100,000 packets. The FTP server employs an infinite FTP flow, which sends TCP segments through OBS network to the FTP client. The maximum segment size (MSS) of the TCP flow is set to 1040 bytes and the receive windows are set to 10,000 MSS. We assume ACK packets do not experience any drop or assembly delay and packet size of ACKs are set to 40 bytes.

On the other hand, 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 ms. Optical links have 1 Gbps bandwidth and 1 ms propagation delay. Electrical links have 500 Mbps bandwidth and 1 ms propagation delay. Optical burst switches have 5 μs switching time and 200 μs per-hop processing delay.

We also use a second OBS network topology, which is the mesh network shown in Fig. 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 ms propagation delay whereas the optical links between the OBS core nodes have 2.5 ms propagation delays. Each FTP server S_i employs an 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 consider both the cases of static FTP flows, where all flows are active throughout the simulations and dynamic FTP flows where some flows are inactive in some parts of the simulations.

3.1 CWBA with linear 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. TCP goodputs achieved by the timer-based and CWBA algorithms for the linear OBS network topology are compared in Fig. 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.

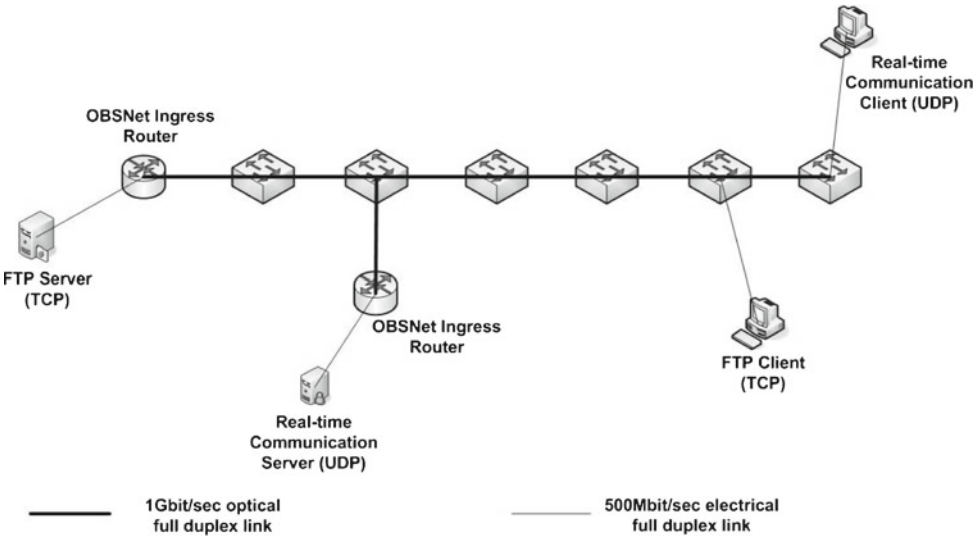

Fig. 1 Linear OBS network topology

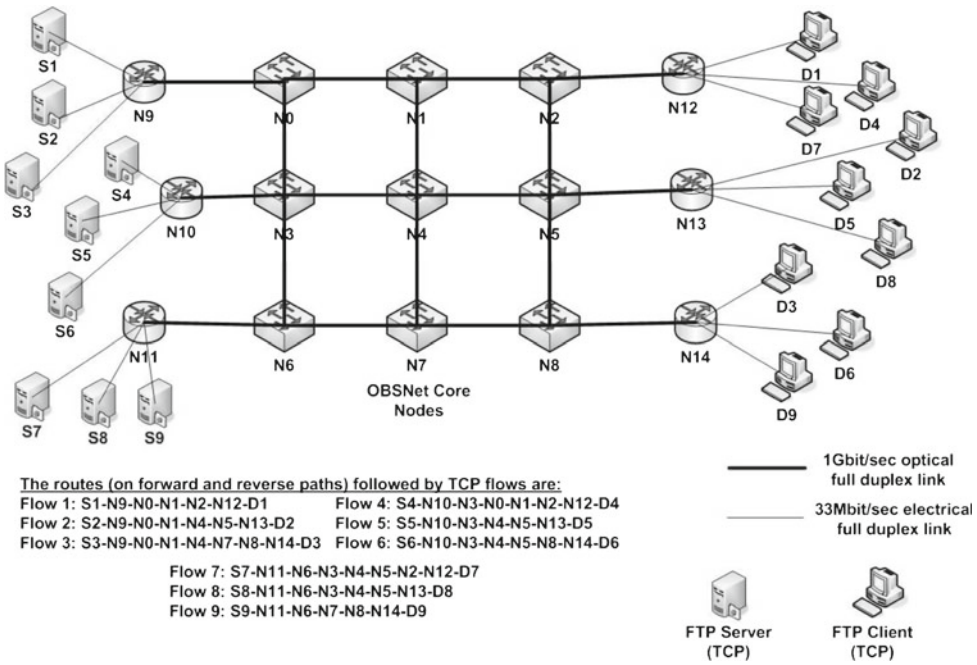

Fig. 2 Mesh OBS network topology

On the other hand, CWBA algorithm achieves about 28% improvement in TCP goodput compared with the throughput obtained when the timer-based burst assembly is used with the optimum value of the timeout and TCP SACK. The performance improvement percentages of CWBA algorithm with respect to the timer-based algorithm with optimum assembly periods for different TCP versions are given in Table 1. Samples of congestion window evolutions are plotted in Figs. 4 and 5 for CWBA and timer-based burst assembly algorithms, respectively. The values of *cwnd* for the timer-based assembly shown in Fig. 5 are obtained using TCP Reno and the optimum assembly delay of 8 ms. The *cwnd* increases approximately 50% faster with CWBA compared with the timer-based assembly in the *slow start* phase. In addition to this, the timer-based assembly algorithm experiences fast retransmission and fast recovery states, whereas with CWBA only timeout loss events occur. It is also observed that in the *congestion avoidance* phase, both algorithms have similar slopes for the linear increase of the congestion window.

3.2 ECWBA with linear topology

The implementation of CWBA algorithm requires some modifications at the TCP layer since some fields in the TCP headers of incoming packets need to be marked accordingly. In order to remove this limitation, we use an estimate of the *cwnd* value at the burst assembler. To this end, we use the

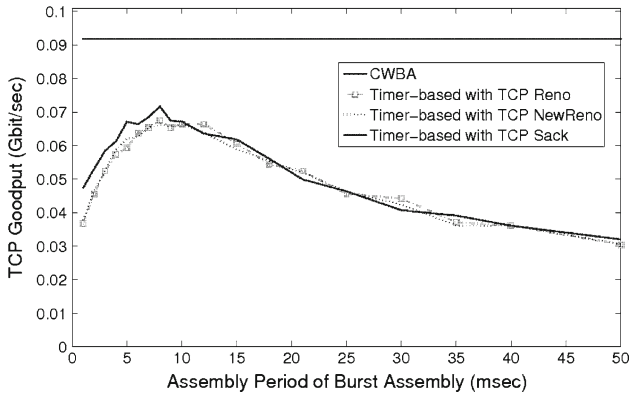

Fig. 3 Performance of timer-based and CWBA algorithms for the linear OBS network topology

Table 1 Goodput improvement of CWBA algorithm for the linear OBS network topology

Reference algorithm (timer-based burst assembly)	Goodput improvement (%)
With TCP Reno	36.01
With TCP NewReno	38.40
With TCP SACK	28.09

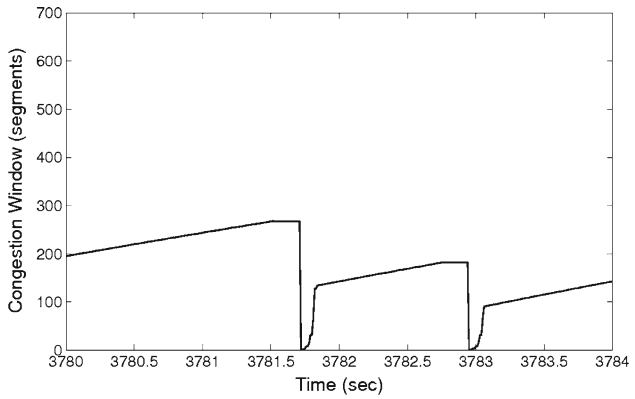

Fig. 4 Congestion window evolution of CWBA algorithm

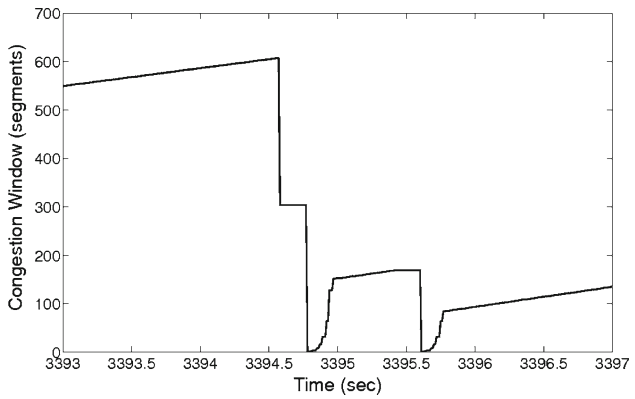

Fig. 5 Congestion window evolution of timer-based assembly algorithm with an assembly delay of 8 ms for TCP Reno

cwnd estimation algorithm described in Sect. 2.3, which creates a replica of the state of the TCP sender's state at the measurement point using passive measurements. 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 using ECWBA at the burst assembler instead of making the exact value of the *cwnd* available at the assembler. The simulations show that there is about 1% reduction in the TCP goodput with ECWBA compared to CWBA. This small degradation is mainly due to the errors in the *cwnd* estimation.

3.3 CWBA with mesh topology

In addition to the linear OBS network topology, we also performed simulations with the mesh OBS network topology shown in Fig. 2. The reported results are obtained by repeating each simulation three times. In this topology, there are nine TCP flows destined to the TCP clients. The route used by each TCP flow is also shown in Fig. 2. In order 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 total goodputs achieved by all TCP flows for CWBA and timer-based assembly algorithms are depicted in Fig. 6. In this scenario, CWBA algorithm achieves 22.36% higher goodput 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 ($AP_{opt} = 26$ ms) are shown in Table 2. AP_{opt} values for the timer-based algorithm and the contribution of each flow to the total goodput for CWBA algorithm are also reported in Table 2. It is observed that all TCP flows increase their respective goodputs with CWBA algorithm.

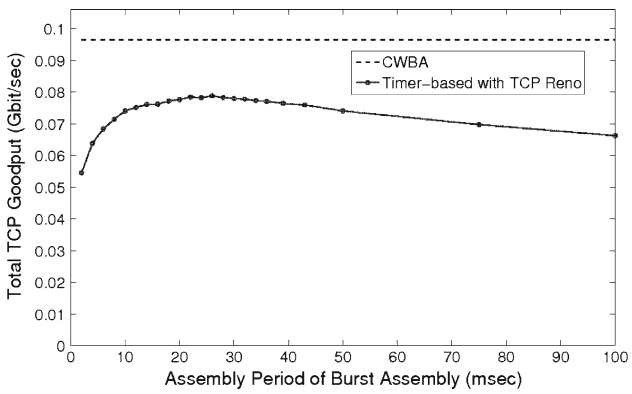

Fig. 6 TCP performance of timer-based and CWBA algorithms in the mesh OBS network topology

Table 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

	Goodput improvement (%)	AP_{opt} (ms)	Contribution (%)
Flow 1	12.02	26	21.05
Flow 2	17.87	18	6.98
Flow 3	21.62	36	10.99
Flow 4	11.43	20	5.48
Flow 5	34.30	32	13.77
Flow 6	13.23	10	7.91
Flow 7	15.63	16	6.62
Flow 8	6.67	50	9.68
Flow 9	19.22	30	17.52

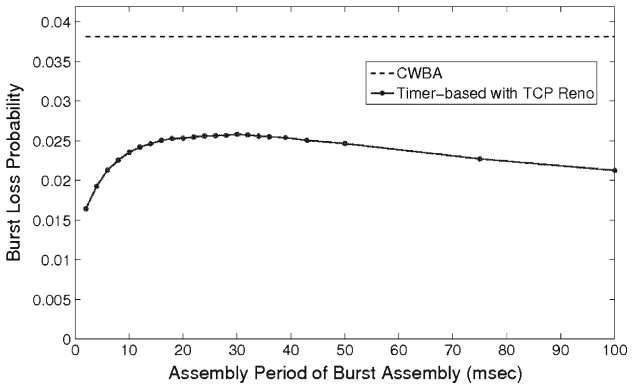

Fig. 7 Burst loss probabilities for timer-based and CWBA algorithms in the mesh OBS network topology

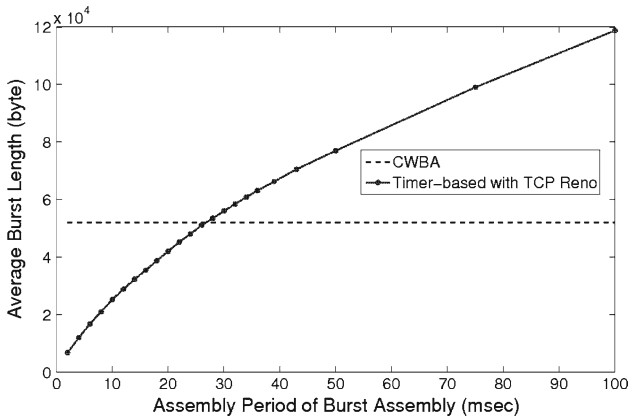

Fig. 8 Average burst lengths for timer-based and CWBA algorithms in the mesh OBS network topology

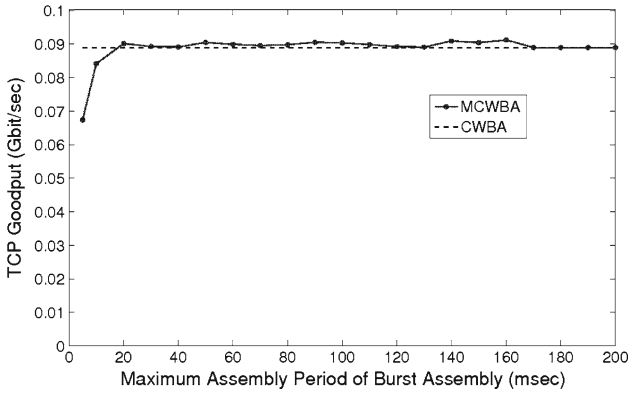

Fig. 9 TCP performance of CWBA and MCWBA algorithms (average of 5 simulations) in the linear OBS network topology

The burst loss probabilities and the average burst lengths for the same scenario are given in Figs. 7 and 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 decreasing contention between bursts. Meanwhile, the burst loss probability for CWBA is higher than the loss rate for the timer-based algorithm. As a conclusion, the simulation results show that the performance gain of CWBA algorithm is mainly due to the reduction in the assembly delay.

3.4 MCWBA with linear topology

MCWBA simulations are performed using the topology in Fig. 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 plotted in Fig. 9. We observe that MCWBA enhances the average per-flow goodput by 2.54% with respect to CWBA.

In order to demonstrate the effects of different assembly algorithms on the performances of individual TCP flows, TCP goodputs corresponding to two different TCP flows are shown in Figs. 10 and 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.

It is observed from the goodput plots in Figs. 9, 10 and 11 that 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 of the largest burst be T_{trans}, switching time of OBS switches be T_{sw} and the per-hop processing delay be T_{hprc}. When the assembly period is larger than $T_{\max} = T_{\text{trans}} + T_{\text{sw}} + T_{\text{hprc}}$, MCWBA algorithm converges to CWBA algorithm. In our simulations with the linear topology, $T_{\max}$ can be calculated as $T_{\max} = 166.6\,\text{ms}$.

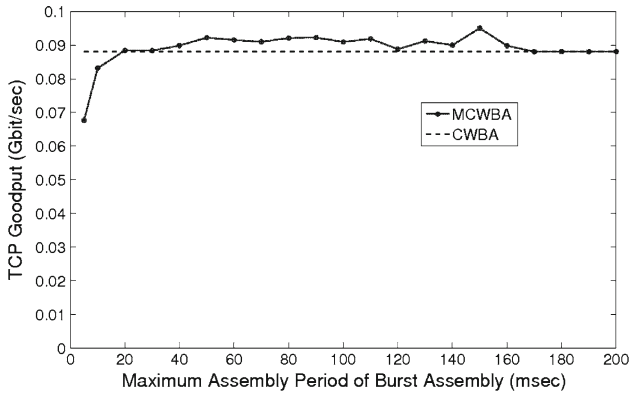

Fig. 10 TCP performance of CWBA and MCWBA algorithms (optimistic case) in the linear OBS network topology

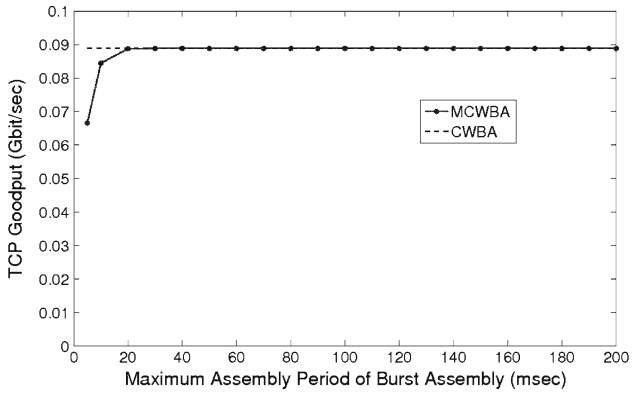

Fig. 11 TCP performance of CWBA and MCWBA algorithms (pessimistic case) in the linear OBS network topology

We conclude that with a proper choice of the assembly period, MCWBA algorithm performs better than the existing assembly algorithms and it puts an upper bound on the maximum delay experienced by a packet.

3.5 MCWBA with mesh topology

MCWBA and CWBA algorithms are compared also using the mesh topology given in Fig. 2. The total goodputs achieved by all TCP flows for both assembly algorithms are depicted in Fig. 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. 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 for CWBA are reported in Table 3.

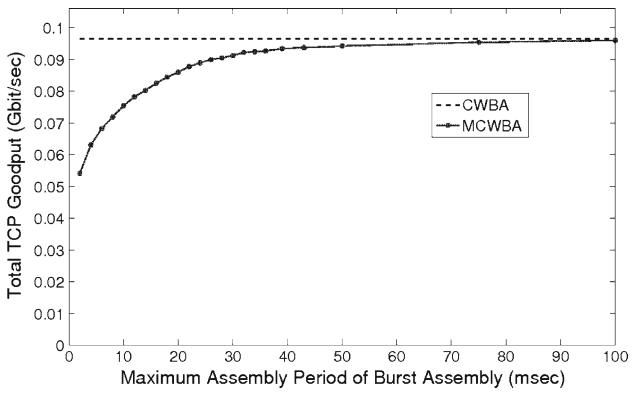

Fig. 12 TCP performance of CWBA and MCWBA algorithms for the mesh OBS network topology

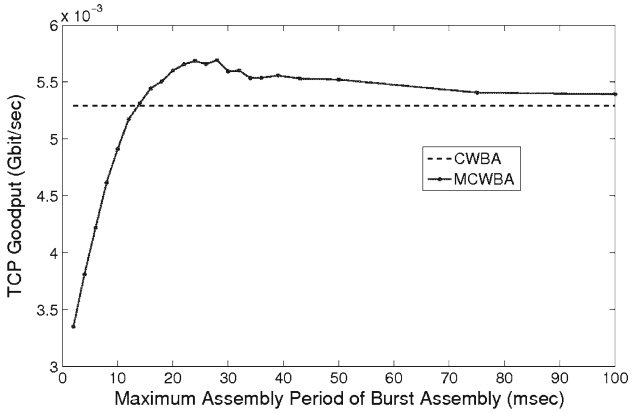

Fig. 13 TCP performance of CWBA and MCWBA algorithms for the 4th TCP flow for the mesh OBS network topology

Table 3 Performance gain of MCWBA compared with CWBA for each TCP flow in the mesh OBS network topology

	Goodput improvement (%)	AP_{opt} (ms)	Contribution (%)
Flow 1	1.60	100	21.05
Flow 2	−4.03	75	6.98
Flow 3	1.47	75	10.99
Flow 4	7.91	28	5.48
Flow 5	−0.52	100	13.77
Flow 6	3.66	28	7.91
Flow 7	0.41	43	6.62
Flow 8	2.52	100	9.68
Flow 9	0.24	100	17.52

The burst loss probabilities of the fourth TCP flow and average loss probability of all flows for the mesh topology are depicted in Fig. 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

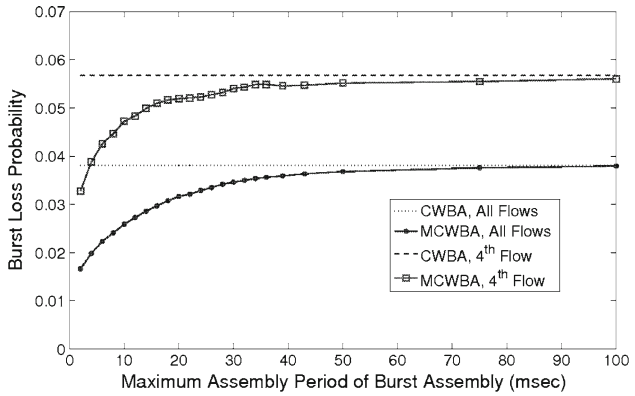

Fig. 14 Burst loss probabilities for CWBA and MCWBA algorithms in the mesh OBS network topology

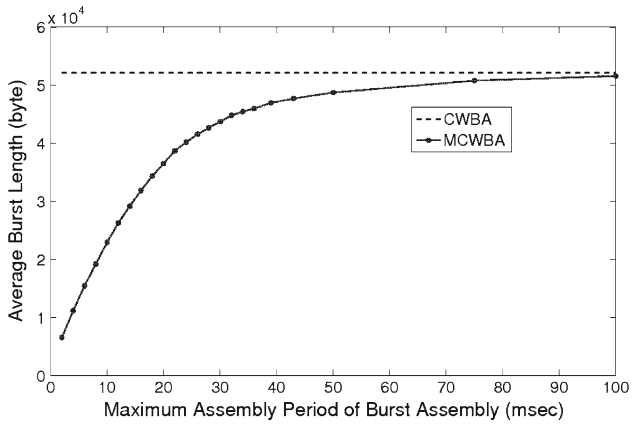

Fig. 15 Average burst lengths for CWBA and MCWBA algorithms in the mesh OBS network topology

than the other flows. Therefore, for the fourth flow small assembly periods yield 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.

In addition to the burst loss probabilities, the average burst lengths of all TCP flows for the mesh topology are shown in Fig. 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 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.

3.6 CWBA with mesh topology and dynamic flows

In the final set of the simulations, we used the mesh OBS topology, where nine TCP flows are dynamically switched on and off during the simulations. Flows 1, 5 and 8 are active for the first half of the simulation ($t \in [0, 5000)$ seconds) whereas Flows 2, 4 and 9 are active for the second half of the simulation ($t \in [5000, 10000)$ seconds) and Flows 3, 6 and 7 are active throughout the simulations. Results are obtained by repeating each simulation three times and then taking the average of these runs. The average total goodput achieved by all TCP flows for both assembly algorithms at the end of the simulation are given in Fig. 16. In this scenario, CWBA algorithm performs 30.5% better than the timer-based algorithm with the optimum assembly period ($AP_{opt} = 28$ ms) on the average. The performance gain for individiual TCP flows are given in Table 4. AP_{opt} values for

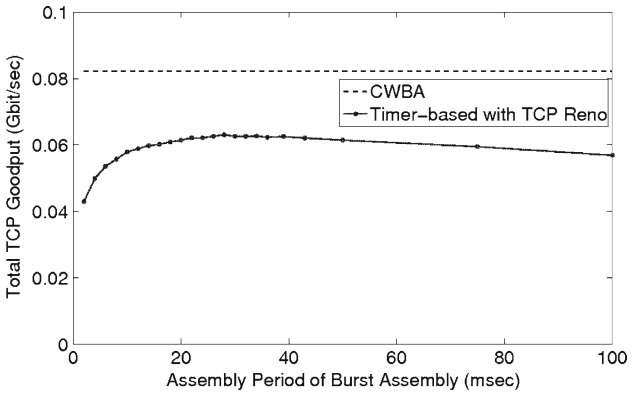

Fig. 16 TCP performance of timer-based and CWBA algorithms in the mesh OBS network topology with dynamic flows ($t \in [0, 10000)$ s)

Table 4 Performance gain of CWBA to timer-based assembly algorithm with the optimum assembly period in the mesh OBS network topology for each dynamic TCP flow ($t \in [0, 10000)$ s)

	Goodput improvement (%)	AP_{opt} (ms)	Contribution (%)
Flow 1	−46.59	28	7.94
Flow 2	55.61	30	7.76
Flow 3	68.19	75	30.27
Flow 4	−10.85	20	3.20
Flow 5	−26.55	28	5.84
Flow 6	−99.39	10	0.06
Flow 7	83.20	20	16.92
Flow 8	108.76	75	12.40
Flow 9	75.60	32	15.61

timer-based algorithm and the contribution of each flow on the total goodput for CWBA algorithm are also reported. The performance gain is much higher than the average especially for TCP flows that individually contribute more than 10% of the total goodput.

In the first half of the simulation, the average of the total goodput achieved by all active TCP flows for both assembly algorithms are given in Fig. 17. In this case, CWBA algorithm performs 16.99% better than the timer-based algorithm with the optimum assembly period ($AP_{opt} = 28$ ms) on the average. The performance gain for each TCP flow, AP_{opt} values for timer-based algorithm and the contribution of each flow on the total goodput for CWBA algorithm are given in Table 5 for the period [0, 5000) seconds. The performance gain achieved by CWBA algorithm is 173.89% for Flow 7, which contributes around 17% of the total goodput and travels through one of the longest routes in the topology.

The average total goodput achieved by all active TCP flows with both assembly algorithms for the second half of the simulation are given in Fig. 18. In this case, CWBA algorithm performs 49.99% better than the timer-based algorithm with the optimum assembly period ($AP_{opt} = 28$ ms) on the average. The performance gain for each TCP flow, AP_{opt} values for timer-based algorithm and also the contribution of each flow on the total goodput for CWBA algorithm are given in Table 6 for the period [5000, 10000) seconds. Again, CWBA performs much better than the timer-based assembler in terms of TCP goodput, especially for the flows that have higher goodputs.

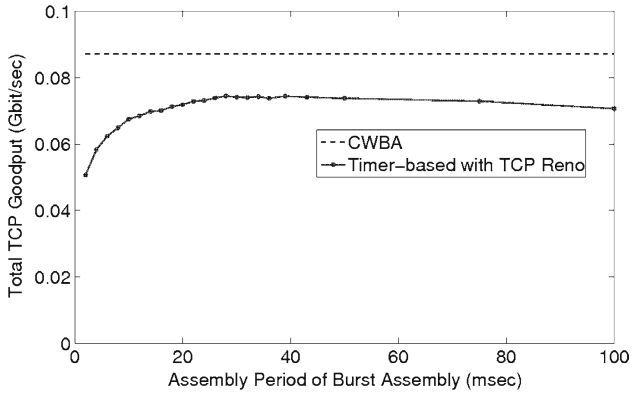

Fig. 17 TCP performance of timer-based and CWBA algorithms in the mesh OBS network topology with dynamic flows ($t \in [0, 5000)$ s)

Table 5 Performance gain of CWBA to timer-based assembly algorithm with the optimum assembly period in the mesh OBS network topology for each dynamic TCP flow ($t \in [0, 5000)$ s)

	Goodput improvement (%)	AP_{opt} (ms)	Contribution (%)
Flow 1	−46.59	28	15.00
Flow 2	–	–	–
Flow 3	35.93	75	33.41
Flow 4	–	–	–
Flow 5	−26.55	28	11.03
Flow 6	−99.00	8	0.11
Flow 7	173.89	14	17.04
Flow 8	108.76	75	23.42
Flow 9	–	–	–

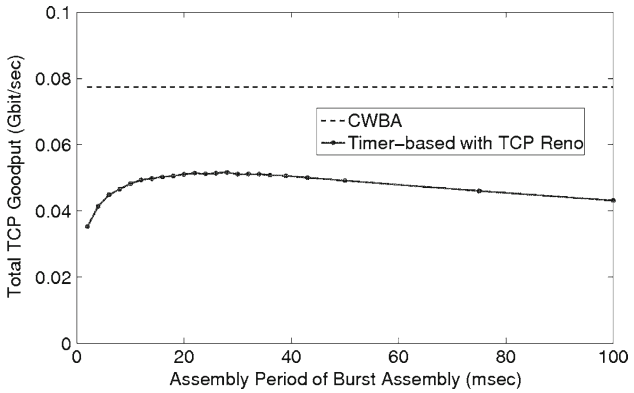

Fig. 18 TCP performance of timer-based and CWBA algorithms in the mesh OBS network topology with dynamic flows ($t \in [5000, 10000)$ s)

Table 6 Performance gain of CWBA to timer-based assembly algorithm with the optimum assembly period in the mesh OBS network topology for each dynamic TCP flow ($t \in [5000, 10000)$ s)

	Goodput improvement (%)	AP_{opt} (ms)	Contribution
Flow 1	–	–	–
Flow 2	55.61	30	16.50
Flow 3	128.85	30	26.74
Flow 4	−10.85	20	6.81
Flow 5	–	–	–
Flow 6	−99.98	10	0.0015
Flow 7	24.5	43	16.79
Flow 8	–	–	–
Flow 9	75.6	32	33.16

4 Conclusion

In this paper, we proposed two new adaptive optical burst assembly algorithms for TCP traffic in OBS networks: CWBA and 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 175% in dynamic network topology scenarios. 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.

Acknowledgements The work described in this paper was carried out with the support of the BONE-project ("Building the Future Optical Network in Europe"), a Network of Excellence funded by the European Commission through the 7th ICT-Framework Programme, and by the Scientific and Technological Research Council of Turkey (TUBITAK) under project 104E047.

References

[1] Yoo, M., Qiao, C.: Optical burst switching (OBS)—a new paradigm for an optical internet. J. High-Speed Netw. **8**(1), 69–84 (1999)
[2] Battestilli, T., Perros, H.: An introduction to optical burst switching. IEEE Commun. Mag. **41**(8), S10–S15 (2003).
[3] Chen, Y., Qiao, C., Yu, X.: Optical burst switching: a new area in optical networking research. IEEE Netw. Mag. **18**, 16–23 (2004)
[4] Cao, X., Li, J., Cao, X., Chen, Y., Qiao, C.: Traffic statistics and performance evaluation in optical burst switched networks. IEEE/OSA J. Lightwave Technol. **22**(12), 2722–2738 (2004)
[5] Cao, X., Li, J., Chen, Y., Qiao,C.: Assembling TCP/IP packets in optical burst switched networks. In: Proceedings of IEEE GLOBECOM'02, pp. 2808–2812 (2002)
[6] Choi, J.Y., Vu, H.L., Cameron, C.W., Zukerman, M., Kang, M.: The effect of burst assembly on performance of optical burst switched networks. Lect. Notes Comput. Sci. **3090**, 729–739 (2004)
[7] Gowda, S., Shenai, R.K., Sivalingam, K.M., Cankaya, H.C.: Performance evaluation of TCP over optical burst-switched (OBS) WDM networks. In: Proceedings of IEEE ICC'03, pp. 1433–1437 (2003)
[8] Oh, S., Hong, H.H., Kang, M.: A data burst assembly algorithm in optical burst switching networks. ETRI J. **24**(4), 311–322 (2002)
[9] Kantarci, B., Oktug, S.: Adaptive threshold based burst assembly in OBS Networks. In: Proceedings of Canadian Conference on Electrical and Computer Engineering (CCECE 2006), pp. 485–488 (2006)
[10] Kantarci, B., Oktug, S.: Path loss rate driven burst assembly in OBS networks. In: Lecture Notes in Computer Science, Proceedings of ISCIS, vol. 4263, pp. 483–492 (2006)
[11] Sideri, A., Varvarigos, E.A.: New assembly techniques for optical burst switched networks based on traffic prediction. In: Lecture Notes in Computer Science, Proceedings of ONDM, vol. 4534, pp. 358–367 (2007)
[12] Venkatesh, T., Sujatha, T.L., Murthy, C.S.R.: A novel burst assembly algorithm for optical burst switched networks based on learning automata. In: Lecture Notes in Computer Science, Proceedings of ONDM, vol. 4534, pp. 368–377 (2007)
[13] Christodoulopoulos, K., Varvarigos, E., Vlachos, K.: A new burst assembly scheme based on the average packet delay and its performance for TCP traffic. Elsevier Opt. Switch. Netw. **4**(3-4), 200–212 (2007)
[14] Ramantas, K., Vlachos, K., Gonzálezde Dios, Ó., Raffaelli, C.: TCP traffic analysis for timer-based burstifiers in OBS networks. In: Lecture Notes in Computer Science, Proceedings of ONDM, vol. 4534, pp. 176–185 (2007)
[15] Zhou, J., Wu, J., Lin, J.: Improvement of TCP performance over optical burst switching networks. In: Lecture Notes in Computer Science, Proceedings of ONDM, vol. 4534, pp. 194–200 (2007)
[16] Peng, S., Li, Z., Wu, X., Xu, A.: TCP window based dynamic assembly period in optical burst switching network. In: Proceedings of IEEE ICC'07, pp. 2365–2370 (2007)
[17] Battestilli, T., Perros, H.: Optical burst switching: a survey. Tech. rep. TR-2002-10, NC State University, Computer Science Department (2002)
[18] Gurel, G., Alparslan, O., Karasan, E.: nOBS: an ns2 based simulation tool for TCP performance evaluation in OBS networks. Ann. Telecommun. **62**(5–6), 618–637 (2007)
[19] Network Simulator 2. developed by L. Berkeley Network Laboratory and University of California Berkeley, http://www.isi.edu/nsnam/ns.
[20] Gurel, G., Karasan, E.: Effect of number of burst assemblers on TCP performance in optical burst switching networks. In: Proceedings of IEEE BROADNETS'06, pp. 1–7 (2006)
[21] Yu, X., Qiao, C., Liu, Y., Towsley D.: Performance evaluation of TCP implementations in OBS networks. Tech. Rep. 2003-13, CSE Department, SUNY, Buffalo, NY (2003)
[22] Wu, J., El-Ocla, H.: TCP congestion avoidance model with congestive loss. In: Proceedings of IEEE ICON'04, pp. 3–8 (2004)
[23] Ming-Chit, I.T., Jinsong, D., Wang, W.: Improving TCP performance over asymmetric networks. ACM Comput. Commun. Rev. **30**(3), 45–54 (2000)
[24] Liao, H., Zhang, Q., Zhu, W., Zhang, Y.-Q.: A robust TCP/IP header compression scheme for wireless networks. In: Proceedings of IEEE International Conference on 3G Wireless and Beyond (2001)
[25] Lance, R., Frommer, I.: Round-trip time inference via passive monitoring. ACM SIGMETRICS PER **33**(3), 32–38 (2005)
[26] Dryna, M.: Network tomography tools. M. S. thesis, Technische Universität München Fakultät für Informatik and Institut Eurécom Sophia-Antipolis (2005)
[27] Jaiswal, S., Iannaccone, G., Diot, C., Kurose, J., Towsley, D.: Inferring TCP connection characteristics through passive measurements. In: Proceedings of IEEE INFOCOM'04, vol. 3, pp. 1582–1592 (2004)

Author Biographies

Seckin Ozsarac received B.S. and M.S. degrees in electrical and electronics engineering from Bilkent University, Ankara, Turkey in 2006 and 2008. He is currently working at TUBITAK-UEKAE/ILTAREN.

Ezhan Karasan received B.S. degree from Middle East Technical University, Ankara, Turkey, M.S. degree from Bilkent University, Ankara, Turkey, and Ph.D. degree from Rutgers University, Piscataway, New Jersey, USA, all in electrical engineering, in 1987, 1990, and 1995, respectively. During 1995-1996, he was a post-doctorate researcher at Bell Labs, Holmdel, New Jersey, USA. From 1996 to 1998, he was a Senior Technical Staff Member in the Lightwave Networks Research Department at AT&T Labs-Research, Red Bank, New Jersey, USA. He has been with the Department of Electrical and Electronics Engineering at Bilkent University since 1998, where he is currently an associate professor. Dr. Karasan is a member of the Editorial Board of Optical Switching and Networking journal. He is the recipient of 2004 Young Scientist Award from Turkish Scientific and Technical Research Council (TUBITAK), 2005 Young Scientist Award from Mustafa Parlar Foundation and Career Grant from TUBITAK in 2004. Dr. Karasan received a fellowship from NATO Science Scholarship Program for overseas studies in 1991–1994. Dr. Karasan has been participating in FP6-IST Network of Excellence (NoE) e-Photon/ONe+ and FP7-IST NoE BONE projects. His current research interests are in the application of optimization and performance analysis tools for the design, engineering and analysis of optical networks and wireless ad hoc/mesh/sensor networks.