Photon Netw Commun (2010) 20:120–130
DOI 10.1007/s11107-010-0252-4

Dynamic threshold-based assembly algorithms for optical burst switching networks subject to burst rate constraints

Mehmet Altan Toksöz · Nail Akar

Received: 29 July 2009 / Accepted: 25 March 2010 / Published online: 17 April 2010
© Springer Science+Business Media, LLC 2010

Abstract Control plane load stems from burst control packets which need to be transmitted end-to-end over the control channel and further processed at core nodes of an optical burst switching (OBS) network for reserving resources in advance for an upcoming burst. Burst assembly algorithms are generally designed without taking into consideration the control plane load they lead to. In this study, we propose traffic-adaptive burst assembly algorithms that attempt to minimize the average burst assembly delay subject to burst rate constraints and hence limit the control plane load. The algorithms we propose are simple to implement and we show using synthetic and real traffic traces that they perform substantially better than the usual timer-based schemes.

Keywords Optical burst switching · Burst assembly delay · Burst assembly algorithms

1 Introduction

Optical burst switching (OBS) has been receiving increasing attention as an alternative transport architecture for the next-generation optical Internet in academia and also in industry [9,12,14]. There are several features of OBS that make it a viable technology. First, in OBS, data travels through the network in the form of relatively long bursts and all-optically. A number of client packets are assembled into a data burst at the edge of an OBS network while the following are taken into consideration: (i) increasing burst lengths helps relax optical switching-speed requirements, (ii) reducing burst lengths also reduces delays stemming from burst assembly. A second principle of OBS is the separation of the control and data planes where the data plane is all-optical but the control plane can be optical-electronic in the sense that control packets are processed electronically at the core nodes. Once a data burst is formed at the edge device, the ingress node prepares a control message on behalf of the data burst and transmits it in the form of a burst control packet (BCP) over the control plane towards the egress node. The BCP carries information about the data burst, such as its length, destination, arrival time, etc. A receipt of a BCP by a core node initiates a configuration of the node by means of reserving resources for the burst when available. On the other hand, the data burst is transmitted over the data plane after an offset time which has to be at least as long as the sum of the per-hop processing times that the corresponding BCP will encounter. In a typical OBS network with no buffers, the end-to-end delay of a single packet is composed of a fixed propagation delay and the sum of the offset time and the burst assembly delay, the minimization of the latter forming the scope of this study.

Various burst assembly algorithms have been proposed to aggregate a number of client packets (such as IP packets) into data bursts. Typically, an ingress node maintains per-destination queues to store client packets awaiting burstification that are destined for a specific destination. Multiple instances of a burst assembly algorithm are run for each of these queues which decide when the packets in the queue should be aggregated into a burst and sent out. Other variations are also possible in which multiple queues are maintained for each destination, one for each QoS-class and different burst assembly algorithms may be run for each of these queues. Such scenarios are left outside the scope of this article.

M. A. Toksöz · N. Akar (✉)
Electrical and Electronics Engineering Department,
Bilkent University, Ankara, Turkey
e-mail: akar@ee.bilkent.edu.tr

M. A. Toksöz
e-mail: altan@ee.bilkent.edu.tr

Four classes of burst assembly algorithms are available in the literature, namely timer-based, size-based, hybrid (timer- and size-based), and dynamic threshold-based algorithms. In timer-based burst assembly [5], a timer is started once a client packet arrives at an empty burst assembly buffer. This timer expires after a period of duration T (in units of s) by which time all packets awaiting in the burst assembly buffer are aggregated into a burst and sent out. The timer parameter T is typically chosen as the largest allowable delay due to burstification. Moreover, a lower burst length $B_{\min}$ (in units of bytes) can also be imposed so that padding is used if the number of bytes awaiting in the buffer upon timer expiration is less than $B_{\min}$. The second class of algorithms are size-based and when the assembly buffer size reaches or exceeds a size parameter B then all packets in the buffer are aggregated into a burst [15]. Clearly, B should be set to a value larger than the lower limit $B_{\min}$. However, these two classes of burst assembly algorithms have their problems of their own. Size-based algorithms suffer from excessive delays especially when the traffic load is light. On the other hand, under heavy traffic load, timer-based algorithms experience a longer average delay than size-based algorithms. The third class of algorithms, namely hybrid timer- and size-based algorithms, keep track of the assembly buffer occupancy, as well as the time since the arrival of the first packet into the assembly buffer. A representative algorithm in this class is proposed in [16] in which an upper burst length limit $B_{\max}$ (in units of bytes) is imposed on the pure timer-based scheme. In this proposal, if the buffer occupancy is to exceed $B_{\max}$ before the timer expires, a portion of the awaiting packets are aggregated into a burst immediately without having to wait for the timer to expire. The final class of algorithms are based on the use of dynamic thresholds, where either the timer parameter T or the size parameter B or both are adjusted dynamically [2,11]. Recently, various methods using dynamic thresholds have been proposed in [4,8,13].

The assumptions we have for the burst assembly problem studied in the current article are given below:

(a) We focus on burst assembly algorithms whose average burst generation rates (both short- and long-term rates) are upper bounded by a desired burst rate parameter called β (in units of bursts/s). We have two main goals with this approach. First, β determines the frequency of BCPs traveling on the control channel and by adjusting β, one can control the control plane load in the system and thus limit BCP queueing delays due to processing. Second, a fair comparison of two burst assembly algorithms is only meaningful when their average burst rates are the same since algorithms with higher burst generation rates are to naturally outperform others in terms of burstification delays.

(b) We impose lower and upper burst length limits $B_{\min}$ and $B_{\max}$ in units of bytes as in [16].

(c) Given the above two constraints, our goal is to devise a burst assembly scheme that minimizes
   - the average packet delay D_{P} which is defined as the average of all packet delays in the assembly buffer, or
   - the average byte delay D_{B} which is defined as the weighted average of all packet delays where the weights are taken to be packet lengths in units of bytes. A burst assembly algorithm that attempts to minimize D_{B} needs to keep track of packet lengths as well.

(d) Finally, we seek a model-free algorithm which is also simple to implement. If the traffic statistics were known, one can obtain an analytical solution as in [7] but generally burstifiers do not have a good understanding of the statistical properties of the traffic streams they need to process. Moreover, traffic is generally unpredictable which leads us to use traffic-adaptive assembly algorithms.

In this study, we mainly focus on the reduction of the delays D_{P} and D_{B} that are caused by the assembly process and we develop two dynamic threshold-based algorithms each of which attempts to minimize one of these two delay parameters under a burst rate constraint β. We then compare our results to those obtained with conventional timer-based schemes under realistic traffic and packet length distribution scenarios. The remainder of this paper is organized as follows. In Sect. 2, we present an overview of existing timer-based and size-based algorithms. The two algorithms we propose are presented in Sect. 3. Section 4 provides numerical results concerning the performance evaluation of existing and proposed algorithms under different traffic scenarios. Finally, Sect. 5 concludes this article.

2 Burst assembly algorithms

In this section, we will first present three conventional burst assembly algorithms, the first two being timer-based, and the third one being size-based. We will then present the two algorithms we propose.

2.1 Timer-based min-length burst assembly

This basic algorithm is given as Algorithm 1. It is called Timer-based min-length burst assembly algorithm, or in short *timer-min* since the algorithm is timer-based and also the minimum burst length limit is enforced. In this algorithm, the inter-burst time is fixed to the timer threshold T which will be set to $1/\beta$. The worst case delay then equals T and assuming packet arrivals for burst i occur uniformly in the interval $((i-1)T, iT)$, the average packet delay is $T/2 = 1/(2\beta)$. This algorithm does not employ an upper limit $B_{\max}$ on burst lengths. The next algorithm attempts to modify the current one by imposing an upper burst length limit.

Algorithm 1 *timer-min*

```
PARAMETERS:
$t$: timer value
$T$: assembly time window
$i$: burst index
$p_i(t)$: data accumulated for the $i$-th burst at timer value $t$ (bytes)
$B_{\min}$: lower burst length limit (bytes)

ALGORITHM
  $t \Leftarrow 0$ {initialize timer to zero}
  if $t = T$ then
    if $p_i(t) \geq B_{\min}$ then
      $p_i(t) \Leftarrow 0$ {send $p_i(t)$ as burst $i$ immediately}
      $i \Leftarrow i + 1$ {increase burst counter}
      $t \Leftarrow 0$ {reset timer}
    else
      $p_i(t) \Leftarrow B_{\min}$ {increase the data size to $B_{\min}$ with padding}
      $p_i(t) \Leftarrow 0$ {send $p_i(t)$ as burst $i$ immediately}
      $i \Leftarrow i + 1$ {increase burst counter}
      $t \Leftarrow 0$ {reset timer}
    end if
  end if
```

2.2 Timer-based min-max-length burst assembly

This modified algorithm is given as Algorithm 2. It is called Timer-based min-max-length burst assembly algorithm, or in short *timer-min-max*, since the upper burst length limit $B_{\max}$ is also imposed. In this algorithm, when the data accumulated for the i-th burst at time t, denoted by $p_i(t)$, at the epoch of timer expiration exceeds $B_{\max}$ then a maximum number of packets whose packet length sum does not exceed $B_{\max}$ is sent out as burst i. Let $\hat{B}_{\max}$ denote the number of bytes sent out. The remaining packets in the burst assembly buffer wait for the next opportunity. In both timer-based algorithms, a decision to assemble is made synchronously without paying attention to the assembly buffer content. Worst case delays are bounded when $B_{\max} \to \infty$ and the burst rate requirement β is inherently taken care of by setting $T = 1/\beta$. One of the main goals of this study is to explore alternative methods that would potentially benefit from asynchronous burst assembly in terms of either average packet or byte delays.

2.3 Fixed threshold-based burst assembly

Assume that the average packet arrival rate to the assembly buffer is known and is denoted by λ. Let us assume $b = \lambda/\beta$

Algorithm 2 *timer-min-max*

```
PARAMETERS:
$t$: timer value
$T$: assembly time window
$i$: burst index
$p_i(t)$: data accumulated for the $i$-th burst at timer value $t$ (bytes)
$B_{\min}$: lower burst length limit (bytes)
$B_{\max}$: upper burst length limit (bytes)

ALGORITHM
  $t \Leftarrow 0$ {initialize timer to zero}
  if $t = T$ then
    if $p_i(t) < B_{\min}$ then
      $p_i(t) \Leftarrow B_{\min}$ {increase the data size to $b$ with padding}
      $p_i(t) \Leftarrow 0$ {send $p_i(t)$ as burst $i$ immediately}
      $i \Leftarrow i + 1$ {increase burst counter}
      $t \Leftarrow 0$ {reset timer}
    else if $p_i(t) \geq B_{\min}$ and $p_i(t) < B_{\max}$ then
      $p_i(t) \Leftarrow 0$ {send $p_i(t)$ as burst $i$ immediately}
      $i \Leftarrow i + 1$ {increase burst counter}
      $t \Leftarrow 0$ {reset timer}
    else
      $p_i(t) \Leftarrow p_i(t) - \hat{B}_{\max}$ {send $\hat{B}_{\max}$ bytes as burst $i$ immediately}
      $i \Leftarrow i + 1$ {increase burst counter}
      $t \Leftarrow 0$ {reset timer}
    end if
  end if
```

is an integer. We can then use a burst assembly algorithm that generates a burst every time b packets are accumulated in the buffer. This strategy ensures a burst generation rate of β. This assembly method will be referred to as *fixed-threshold*. It is then crucial to know whether this policy is optimal. Let us assume renewal inter-packet arrival times with mean α. Let us use an arbitrary probabilistic policy that assembles when b_i packets are present with probability p_i, $1 \leq i \leq N$. To enforce a burst generation rate of β, we should have $\sum_{i=1}^{N} b_i p_i = b$. An arbitrary packet will then belong to a burst with length b_i with probability $\frac{b_i p_i}{b}$, $1 \leq i \leq N$. The average packet delay then becomes

$$D_P = \frac{\alpha}{2b} \sum_{i=1}^{N} p_i b_i (b_i - 1) \qquad (1)$$

It can be shown that the average delay is minimized with a deterministic policy $N = 1$ that generates a burst every time b packets are accumulated in the buffer. In this case

$$D_P = \frac{\alpha(b-1)}{2} \qquad (2)$$

which provides an expression for the optimum average packet delay. For instance, if λ is 50,000 packets/s and β is 1,000 bursts/s, then an optimal burst assembly policy will be to wait for 50 packets to arrive for burst assembly. It is very likely that the value $b = \lambda/\beta$ may not be an integer. Say the value b is in the form $x + y$ where x is the integer part of b and y is the fractional part where $0 < y < 1$. The optimal policy in this case is one which assembles packets when x packets are accumulated with probability $1 - y$, or when $x+1$ packets are accumulated with probability y. There are several drawbacks of this dynamic threshold-based burst assembly mechanism described above:

- The method is very sensitive to the average packet arrival rate λ; a deviation of the estimate from the actual value will lead to burst generation rates that differ from β.
- When the packet arrival process is a non-renewal process, using a fixed threshold of b packets for burst assembly would generate bursts at a long-term rate of β but over relatively shorter terms, the burst rate constraints can be violated leading to occasional problems on the control plane. For this scenario, a need arises to employ a dynamic-threshold algorithm to keep track of changes in the arrival process so as to maintain the short-term burst rate averages at a desired rate of β as well. This situation appears to worsen with non-stationary traffic.
- When b packets are accumulated, most of these packets can turn out to be relatively large packets making the total length exceed $B_{\max}$. It appears to be very difficult to enforce in this algorithm the upper limit $B_{\max}$ which is in units of bytes. The lower limit can be enforced by padding.
- Since the algorithm keeps track of only the number of packets and not their lengths, this algorithm cannot differentiate between packet and byte delays. If the focus is the minimization of byte delays, then we should resort to a modified algorithm.

Although a fixed-threshold-based burst assembly algorithm has nice theoretical properties, we still seek a method that is model-free, which is simple to implement, and which keeps track of bytes for the purposes of enforcing the lower and upper bandwidth limits as well as the minimization of average byte delay in addition to average packet delay.

3 Proposed burst assembly algorithms

The proposed algorithms we propose do not require any prior information such as the average packet arrival rate or average bit rate. Another strength of the proposed algorithms is their simplicity as compared to other dynamic-threshold algorithms. Next, we present these two algorithms.

3.1 Packet-based dynamic-threshold algorithm for burst assembly

This algorithm (given as Algorithm 3) is an entirely packet-based algorithm and it is referred to as *dyn-threshold-packet* in short. In this algorithm, we keep track of the packet count in the assembly buffer and we aim to minimize the average packet delay due to burstification. The lower and upper burst length limits are given in units of packets and they are denoted by $L_{\min}$ and $L_{\max}$, respectively. We also maintain a counter called *bucket* to indicate the dynamic threshold used in our burst assembly algorithm. Each time a packet, say packet k, arrives at the assembly buffer, the bucket is decremented by β times the inter-arrival time between packets $k-1$ and k. A decision for burst assembly is made only when the current packet count exceeds the bucket value. When an assembly decision is made, the bucket is incremented by one. To enforce lower and upper burst length limits, the bucket is allowed to take values in the interval $[L_{\min}, L_{\max} - 1]$. We have also added an expiration time $T_{\max}$ for a burst to meet the worst case delay requirement. Even if the conditions for a burst are not met in low traffic load, the expiration time mechanism would force the generation of the burst.

Algorithm 3 dyn-threshold-packet

```
PARAMETERS:
$i$: packet index
$j$: burst index
$\beta$: desired burst rate (bursts/s)
$L(i, j)$: data accumulated for the $j$-th burst at the arrival epoch of the $i$-th packet (in units of packets)
$L_{\min}$: lower burst length limit (in units of packets)
$L_{\max}$: upper burst length limit (in units of packets)
$bucket$: dynamic threshold
$t$: timer value
$T_{\max}$: burst expiration time
$t_i$: inter-arrival time between the $(i-1)$st and $i$th packets

ALGORITHM
  if $L(i, j) = 1$ then
    $t \Leftarrow 0$ {if the assembly queue contains 1 packet, start the timer}
  end if
  $bucket \Leftarrow bucket - t_i \beta$ {leak the bucket}
  $bucket \Leftarrow \max(L_{\min}, bucket)$ {enforce lower burst length limit}
  if $L(i, j) \geq bucket$ then
    $L(i, j) \Leftarrow 0$ {send $L(i, j)$ as burst $j$ immediately}
    $bucket \Leftarrow \min(bucket + 1, L_{\max} - 1)$ {update bucket and enforce upper burst length limit}
    $j \Leftarrow j + 1$ {increase burst counter}
    $t \Leftarrow 0$ {reset timer}
  else if $t \geq T_{\max}$ then
    $L(i, j) \Leftarrow \max(L_{\min}, L(i, j))$ {increase the data size to $L_{\min}$ with padding if necessary}
    $L(i, j) \Leftarrow 0$ {send $L(i, j)$ as burst $j$ immediately}
    $j \Leftarrow j + 1$ {increase burst counter}
    $t \Leftarrow 0$ {reset timer}
  end if
```

3.2 Byte-based dynamic threshold algorithm for burst assembly

This algorithm (given as Algorithm 4) is a byte-based algorithm and it is referred to as *dyn-threshold-byte* in short. In this algorithm, we keep track of the byte count in the assembly buffer and we aim to minimize the average byte delay due to burstification. The reason for this is that client packet lengths are variable; short and long packets are to be treated differently since they contribute differently to the overall byte delay. The lower and upper burst length limits are given in units of bytes and they are denoted by $B_{\min}$ and $B_{\max}$, respectively. Similar to the *dyn-threshold-packet* algorithm, we maintain a *bucket* to indicate the dynamic threshold used in our burst assembly algorithm. Each time a packet, say packet k, arrives at the assembly buffer, the bucket is decremented by an amount in direct proportion with the inter-arrival time between packets $k-1$ and k with the constant of proportionality set to $\kappa\beta$. A decision for burst assembly is made only when the current byte count exceeds the bucket value. When an assembly decision is made, the bucket is incremented by κ. The parameter κ is the learning parameter of the system. A large value of κ indicates an algorithm that rapidly tracks changes in incoming traffic. However, when κ is large, it is possible to occasionally deviate from the desired burst rate β. The parameter κ should be chosen by taking into consideration of these two effects. Unless otherwise stated, we use $\kappa = 1{,}000$ in our numerical examples. To enforce lower and upper burst length limits, the bucket is allowed to take values in the interval $[B_{\min}, B_{\max} - P_{\max}]$ where $P_{\max}$ denotes the length of the maximum-sized packet. The expiration time $T_{\max}$ is again used.

Algorithm 4 dyn-threshold-byte

```
PARAMETERS:
$i$: packet index
$j$: burst index
$\beta$: burst rate (bursts/s)
$D(i, j)$: data accumulated for the $j$-th burst at the arrival of the $i$-th packet (bytes)
$B_{\min}$: lower burst length limit (bytes)
$B_{\max}$: upper burst length limit (bytes)
$P_{\max}$: maximum packet length (bytes)
$\kappa$: learning parameter
$bucket$: dynamic threshold
$t$: timer value
$T_{\max}$: burst expiration time
$t_i$: inter-packet time between the $(i-1)$st and $i$th packets

ALGORITHM
  if $D(i, j)$ contains 1 packet then
    $t \Leftarrow 0$ {start the timer}
  end if
  $bucket \Leftarrow bucket - t_i\beta\kappa$ {leak the bucket}
  $bucket \Leftarrow \max(B_{\min}, bucket)$ {enforce lower burst length limit}
  if $D(i, j) \geq bucket$ then
    $D(i, j) \Leftarrow 0$ {send $D(i, j)$ as burst $j$ immediately}
    $bucket \Leftarrow \min(bucket + \kappa, B_{\max} - P_{\max})$ {update bucket and enforce upper burst length limit}
    $j \Leftarrow j + 1$ {increase burst counter}
    $t \Leftarrow 0$ {reset timer}
  else if $t \geq T_{\max}$ then
    $D(i, j) \Leftarrow \max(B_{\min}, D(i, j))$ {increase the data size to $B_{\min}$ with padding if necessary}
    $D(i, j) \Leftarrow 0$ {send $D(i, j)$ as burst $j$ immediately}
    $j \Leftarrow j + 1$ {increase burst counter}
    $t \Leftarrow 0$ {reset timer}
  end if
```

4 Numerical results

We will present our numerical results basically for two different types of traffic scenarios (i) synthetic traffic (ii) real traffic traces. We will use synthetic traffic mainly to show several theoretical properties of the burst assembly algorithms mentioned above.

4.1 Synthetic traffic

We study in this section two synthetic traffic models, the first one being the Poisson traffic model, and the second one being the Markov Modulated Poisson process (MMPP) model [6]. MMPP is not a renewal process but instead a Markov renewal process in which the successive inter-arrival times depend on each other. MMPP-based traffic models capture auto-correlation and they are commonly used in the modeling of Internet traffic [10].

4.1.1 Poisson traffic scenario

We first assume that the input packet traffic is stationary Poisson with arrival rate λ (in units of packets/s). Under the burst rate constraint dictated by β, we can calculate the threshold and average packet delay for the threshold-based algorithms, and the average packet delay for the timer-based algorithms. As stated before, under these assumptions, the fixed threshold which minimizes the average packet delay for the *fixed-threshold* algorithm is given by $b = \lambda/\beta$. Recall that the average packet delay of *fixed-threshold* is given by $D_P = (b-1)/(2\lambda) = 1/(2\beta) - 1/(2\lambda)$. On the other hand, the average packet delay for the *timer-min* algorithm is $1/(2\beta)$ as we mentioned earlier. The term $1/(2\lambda)$ is the reduction in packet delays using a size-based algorithm that has a-priori information on λ. In order to verify the results obtained above and to compare them against the algorithms we propose, we have designed a simulation scenario as given below:

– Packet arrival process is stationary Poisson with rate λ that is varied from 5,000 to 50,000.
– Desired burst rate β is set to 1,000.
– Packet size distribution is taken from Table 1, which uses the traffic traces from [3]. To clarify, the first row of Table 1 suggests that 29.55% of all the packets have lengths (in units of bytes) in the interval $[32, 64)$ and 2171017 such packets are observed. For convenience, in our simulations, we assume that with probability 0.2955,

Table 1 Packet size distribution from [3]

Size range (bytes)	# Packets	Probability
32–64	2171017	0.2955
64–128	2519797	0.2621
128–256	574504	0.0598
256–512	297002	0.0309
512–1024	251686	0.0262
1024–2048	3800020	0.3953

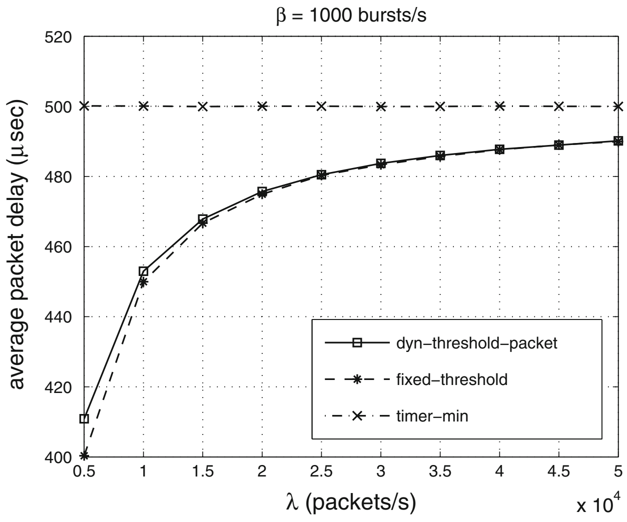

Fig. 1 Average packet delay of the three burst assembly algorithms as a function of the arrival rate λ

an incoming packet has a discrete uniform distribution in the interval [32, 64), with probability 0.2621, it has a discrete uniform distribution in the interval [64, 128), and so on. We believe that our synthetic method of generating packet lengths matches quite well with real traffic traces. Unless otherwise stated, this packet size distribution method will be used throughout the numerical examples used in this paper.
- Simulation length is 1,000 s.
- Lower and upper burst length limits are not enforced.

Figure 1 compares the average packet delay of the three algorithms *timer-min*, *fixed-threshold*, and *dyn-threshold-packet* as a function of the arrival rate λ. As $\lambda \to \infty$, the average packet delay of *fixed-threshold* approaches to that of *timer-min* validating the closed-form expressions stated before. The average packet delay obtained by *dyn-threshold-packet* follows very closely the curve of *fixed-threshold* for all arrival rates. Note that *dyn-threshold-packet* does not assume an a-priori knowledge of the arrival rate λ as *fixed-threshold*. In Fig. 2, we also observe that *dyn-threshold-packet* achieves a burst rate which is very close to β validating the burst rate conformance of bucket-based algorithms.

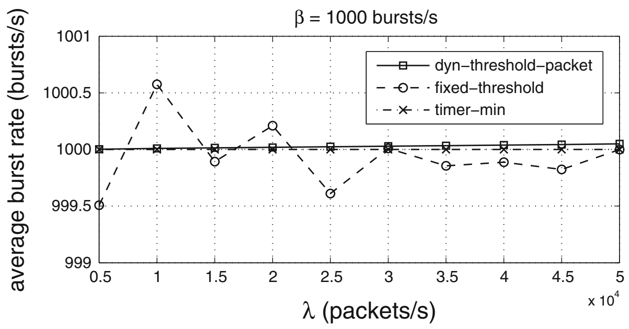

Fig. 2 Average burst rate obtained using the three burst assembly algorithms as a function of the arrival rate λ

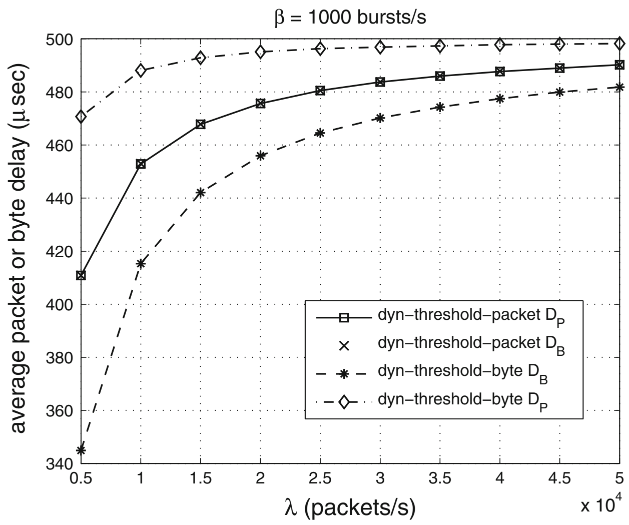

Fig. 3 Average packet and byte delays (D_P and D_B) for the two algorithms *dyn-threshold-packet* and *dyn-threshold-byte* as a function of the arrival rate λ

We propose *dyn-threshold-byte* for the purpose of reducing average byte delays instead of packet delays. Average packet and byte delays (D_P and D_B) for the two algorithms *dyn-threshold-packet* and *dyn-threshold-byte* as a function of arrival rate λ are given in Fig. 3, which shows that the algorithm *dyn-threshold-packet* generates identical byte and packet delays since this algorithm is not aware of packet lengths. On the other hand, the length-aware algorithm *dyn-threshold-byte* substantially reduces D_B. We are led to believe that one should use *dyn-threshold-byte* if the minimization of byte delays are sought.

4.1.2 MMPP traffic scenario

We experiment a non-renewal inter-arrival scenario using synthetic traffic. For this purpose, we use a two-state MMPP to model client packet arrivals to the assembly buffer as shown in Fig 4. In this model, $\lambda_i, i = 1, 2$ denotes the arrival rate at state i. The average state holding time in state i is denoted by T_i. Therefore, the transition rate from

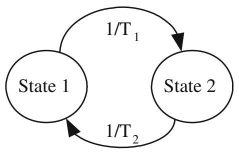

Fig. 4 State diagram of the input traffic modeled by a two-state MMPP

state 1 to state 2 (from state 2 to state 1) in Fig. 4 is $1/T_1$ ($1/T_2$). The average packet arrival rate is denoted by $\lambda = (\lambda_1 T_1 + \lambda_2 T_2)/(T_1 + T_2)$.

The *timer-min* algorithm produces $D_P = 1/2\beta$ irrespective of incoming packet traffic characteristics. The *fixed-threshold* algorithm assumes a-priori information on average arrival rate λ and generates bursts each time $b = \lambda/\beta$ packets are accumulated assuming integer b. The average packet delay for the *fixed-threshold* algorithm can then be written as:

$$D_P = \frac{\left(\frac{\lambda}{\beta} - 1\right)\frac{1}{2\lambda_1}\lambda_1 T_1 + \left(\frac{\lambda}{\beta} - 1\right)\frac{1}{2\lambda_2}\lambda_2 T_2}{\lambda_1 T_1 + \lambda_2 T_2} \tag{3}$$

Let us now use another scheme called *optimum* that is aware of the state which the MMPP is visiting. For the purposes of optimal performance, this scheme generates bursts in state 1 (in state 2) when $b_1 = \lambda_1/\beta$ ($b_2 = \lambda_2/\beta$) packets are accumulated. Here, we again assume b_1 and b_2 are integers. The burst rate of the *optimum* scheme is then equal to β irrespective of which state of MMPP is being visited. The average packet delay for the *optimum* scheme is easy to write:

$$D_P = \frac{\left(\frac{\lambda_1}{\beta} - 1\right)\frac{1}{2\lambda_1}\lambda_1 T_1 + \left(\frac{\lambda_2}{\beta} - 1\right)\frac{1}{2\lambda_2}\lambda_2 T_2}{\lambda_1 T_1 + \lambda_2 T_2} \tag{4}$$

It is not difficult to show that the two expressions in (3) and (4) lead to identical average packet delay D_P which can further be simplified to

$$D_P = \frac{1}{2\beta} - \frac{1}{2\lambda} \tag{5}$$

The second term above characterizes the reduction in average packet delay by using a size-based algorithm as opposed to a timer-based algorithm. Note that this term is identical to that of the Poisson traffic scenario. We therefore conclude that the *fixed-threshold* algorithm provides optimum average packet delay but it suffers from fluctuations in the burst rate. When the actual traffic rate exceeds the mean rate, the burst rate of the *fixed-threshold* method exceeds the desired burst rate β. Similarly, when the actual rate is lower than the mean rate, burst rates are lower than β. On the other hand, the *optimum* scheme produces optimal D_P while maintaining the burst rate at β at all times. However, it is very hard to implement the *optimum* scheme since in this scheme, the traffic model should be entirely available to the burst assembly unit which should also accurately estimate the instantaneous state of the MMPP. In order to study how the proposed algorithms compare to these three algorithms, we experiment a scenario where $T_1 = \gamma t$ and $T_2 = (1 - \gamma)t$ where $t = 10$ s, $0 < \gamma < 1$ and $\lambda_1 = 5{,}000$ and $\lambda_2 = 50{,}000$ packets/s. The lower and upper burst length limits are not enforced in this experiment. We have tested the algorithms for three different values of $\gamma = 0.3, 0.5, 0.7$ for each algorithm. Let b_i^* and β_i^*, $i = 1, 2$ denote the average threshold value (in units of packets) and average burst generation rate (in units of bursts/s) while at state i. We provide b_i^* and β_i^*, $i = 1, 2$ as well as the average packet delay D_P using the *fixed-threshold*, *optimum*, and *dyn-threshold-packet* algorithms as a function of γ in Table 2. Note that the *timer-min* algorithm average delay is fixed at 500 µs for all examples. In the *fixed-threshold* algorithm, the thresholds are fixed irrespective of the state of the MMPP and therefore the burst rates in each state deviate substantially from the desired burst rate although the long-term burst rate is kept approximately at β. The *optimum* scheme employs two separate burst assembly thresholds depending on the MMPP state and burst generation rate can therefore be set to β irrespective of the MMPP state. The average packet delays for these two algorithms are very close to each other as expected (see expression (5)). The proposed *dyn-threshold-packet* algorithm performs very close to the *optimum* method by adjusting properly the assembly thresholds at each state so that the burst generation rate settles at β and its delay performance is very close to the

Table 2 The values b_i^* and β_i^*, $i = 1, 2$ and D_P using the *fixed-threshold*, *optimum*, and *dyn-threshold-packet* algorithms as a function of γ

Algorithm	γ	b_1^*	b_2^*	β_1^*	β_2^*	D_P (µs)
Fixed-threshold	0.3	36.10	36.12	138.49	1384.04	486.26
	0.5	28.13	28.12	177.74	1777.88	482.29
	0.7	21.23	21.22	235.49	2356.44	476.71
Optimum	0.3	5	50	1000.53	999.96	486.53
	0.5	5	50	999.95	999.86	481.70
	0.7	5	50	1000.12	999.69	472.67
Dyn-threshold-packet	0.3	5.08	49.69	985.07	1006.20	486.94
	0.5	5.04	49.56	991.09	1008.89	482.87
	0.7	5.03	49.30	993.67	1014.27	475.46

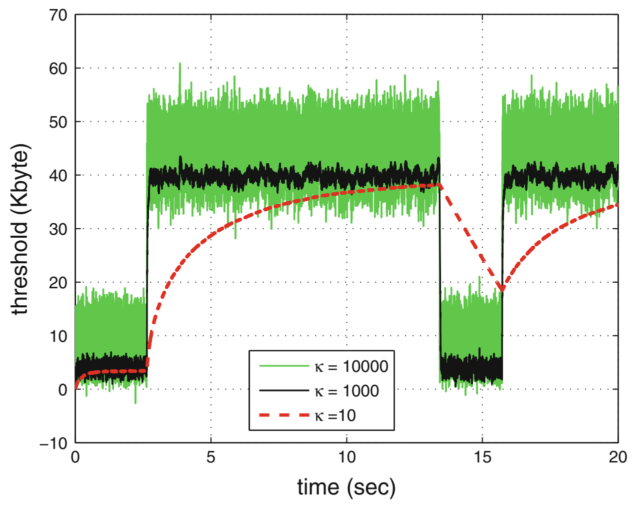

Fig. 5 A 20-s snapshot of the dynamic thresholds of the *dyn-threshold-byte* algorithm with respect to time for different values of κ

size-based algorithms. Despite the difficulty in implementing the *optimum* method, our proposed method is model-free and is very easy to implement.

For *dyn-threshold-byte* algorithm, in order to see the effects of the choice of the learning parameter κ, we plotted the dynamic thresholds as a function of time for various values of κ when γ is set to 0.5 in the previous example. As we see in Fig. 5, for $\kappa = 10$, the dynamic threshold changes slowly despite the abrupt change in the traffic and the algorithm comes short of tracking the thresholds of the *optimum* scheme. For $\kappa = 10{,}000$, on the other hand, change in traffic is captured but at the expense of large-scale fluctuations in the dynamic threshold. We also provide Table 3 which presents the quantities b_i^*, $\beta_i^*, i = 1, 2$ and D_B using the *dyn-threshold-byte* algorithm as a function of κ. It is clear that large-scale fluctuations in the dynamic threshold result in increases in the average byte delay D_B. We conclude that the choice of the learning parameter $\kappa = 1{,}000$ is a reasonable choice since in this case κ is large enough to track rapid changes in traffic and κ is small enough to make sure that fluctuations in the dynamic threshold are reasonably small. We set κ to 1,000 in the remaining numerical studies of the current article.

Table 3 The values b_i^* and $\beta_i^*, i = 1, 2$ and D_B using the *dyn-threshold-byte* algorithm as a function of κ

κ	β_1^*	β_2^*	β	D_B (µs)
1	224.01	1733.19	1002.88	466.72
10	611.88	1364.66	1000.19	467.86
100	935.17	1063.37	1000.00	467.76
1000	992.61	1007.24	1000.00	469.55
10000	999.25	1000.73	1000.00	478.09
30000	999.73	1000.26	1000.00	484.41

4.2 Real traffic traces

In the previous scenarios driven with synthetic traffic, we have shown the basic properties of various burst assembly methods. However, it is also crucial to study the delay performance of the proposed algorithms in case of more realistic traffic scenarios. In this numerical experiment, we focused on only byte delays and not packet delays. For this purpose, we use two different traces taken from a traffic data repository maintained by the measurement and analysis on the WIDE Internet (MAWI) working group of the WIDE Project [3]. We also scale down the inter-arrival times in these traces to generate varying incoming bit rates. While the first trace has a low standard deviation (STD), the latter is quite bursty. For each traffic trace, we use three different values of $\beta = 1000, 2000, 3000$. The lower and upper burst length limits have been enforced in this experiment, i.e., $B_{\min} = 1$ kb and $B_{\max} = 70$ kb. We have studied the performance of the *dyn-threshold-byte* algorithm against the *timer-min* and the *timer-min-max* algorithms. The learning parameter κ is set to 1,000 for *dyn-threshold-byte* and $T_{\max}$ is set to ∞. We have not tested the *fixed-threshold* algorithm in this scenario due to its highly variable burst rates that may not be desirable.

The first trace was obtained from the WIDE backbone at Sample Point B on Jan 1, 2006 at 14:00:00 for a trans-Pacific line with 100 Mbps link speed [3]. The original trace has a duration of 899.76 s, mean rate = 22.33 Mbps, and STD = 1.53 M. Feeding the trace to the burst assembly unit with varying bit rates (by scaling down the inter-arrival times), we have simulated the performance of various burst assembly algorithms. The average byte delays for the three algorithms are given in Fig. 6a–c for three different values of β. Figure 6d gives a minute-long snapshot of the incoming bit rate (scaled 14 times) as a function of time. The trace is pretty smooth similar to a Poisson traffic stream and therefore *timer-min* and *timer-min-max* performed very similarly since the probability that the accumulated number of bytes within a timer expiration period exceeding $B_{\max}$ was negligibly small for this smooth traffic. The results clearly show that the proposed *dyn-threshold-byte* significantly reduces the average byte delay compared to timer-based algorithms especially for lower bit rates. The percentage gain in using our proposed algorithm also increases with β.

We then study the second trace which was obtained again from the WIDE backbone at Sample Point F on Sat Jan 5, 2008 at 14:00:00 for a trans-Pacific line with 150 Mbps link speed [3]. The original trace has a duration of 900.29 s, mean rate = 61.56 Mbps, and STD = 11.67 M. The average byte delays for the three algorithms are given in Fig. 7a–c for three different values of β. Figure 7d gives a 2-min-long snapshot of the incoming bit rate (scaled 7 times) as a function of time. The trace is not as smooth as the previous one and is quite bursty. Therefore, when enforcing

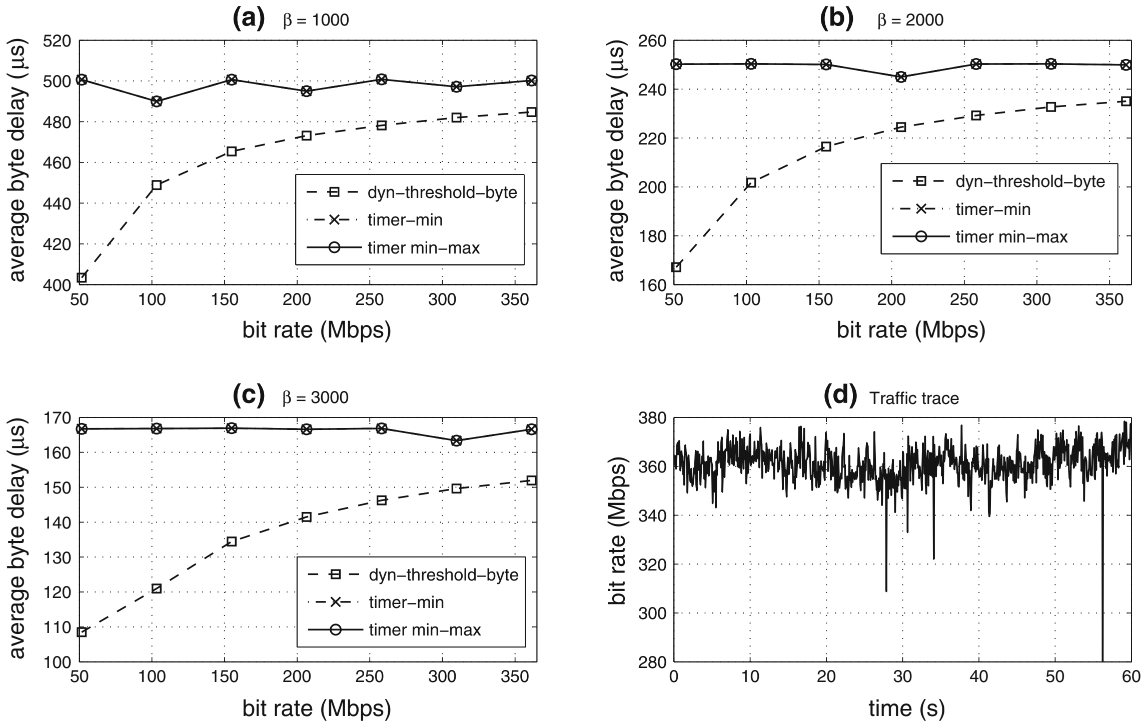

Fig. 6 Average byte delay for the cases **a** $\beta = 1{,}000$ **b** $\beta = 2{,}000$ **c** $\beta = 3{,}000$ using various algorithms for the trace from Sample Point B (2006) whose 1-min snapshot is given in **d**

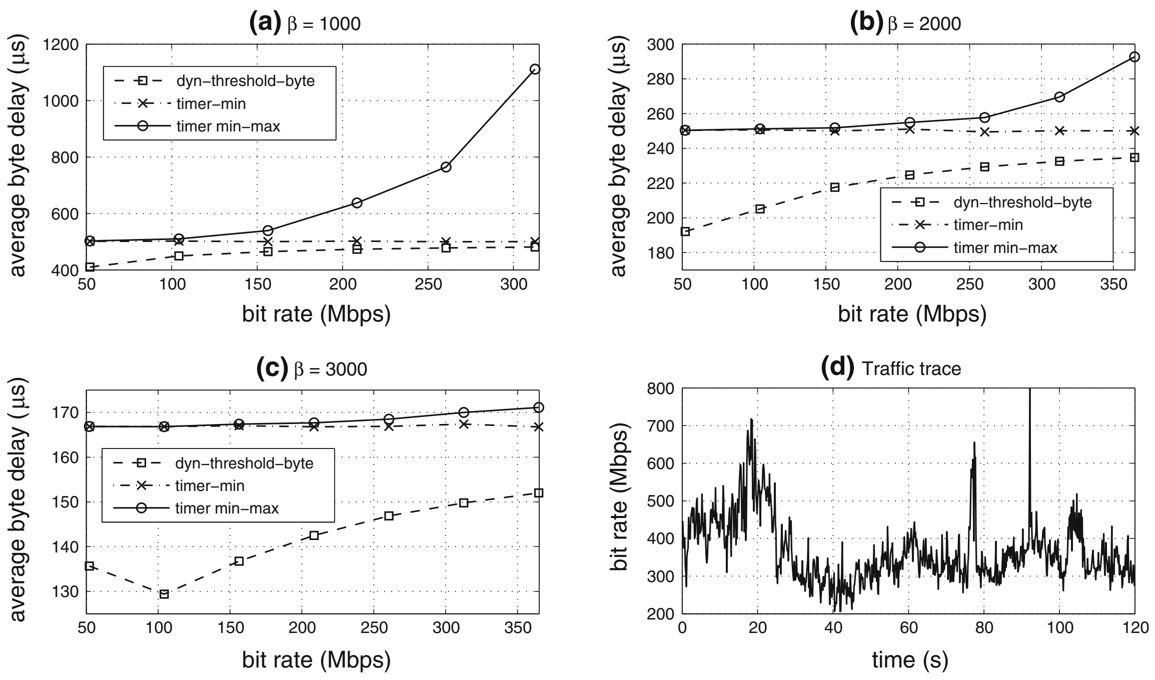

Fig. 7 Average byte delay for the cases **a** $\beta = 1{,}000$ **b** $\beta = 2{,}000$ **c** $\beta = 3{,}000$ using various algorithms for the trace from Sample Point F (2008) whose 2-min snapshot is given in **d**

the upper burst length limit, there were quite a few occasions at which the accumulated number of bytes within a timer expiration period exceeded $B_{\max}$ and some packets had to wait for the next timer expiration epoch when using *timer-min-max*. In this case, the *timer-min-max* performed very poorly compared to the *timer-min* algorithm for which there was no enforcement of $B_{\max}$. As expected, this situation is more evident for relatively lower β. The proposed *dyn-threshold-byte* is shown to significantly reduce the average byte delay D_B compared to both timer-based algorithms especially for lower bit rates and higher β. We also note that *dyn-threshold-byte* not only reduces D_B but also properly enforces the lower and upper burst length limits.

4.3 Loss performance

In the previous numerical studies, we have shown the reductions in average packet or byte delays in the burst assembly buffer using the proposed dynamic-threshold algorithms while enforcing lower and upper burst length limits. However, it is also vital to address the traffic statistics of the bursts fed into the OBS network and their impact on burst loss performance in the OBS network. Recall that the *timer-min* or *timer- min-max* algorithms produce deterministic burst inter-arrival times with variable burst lengths whereas the *fixed-threshold* algorithm generates bursts that have fixed number of packets in them but variable inter-burst times. The proposed algorithms in this article produce both variable inter-burst times and burst lengths. In this section, we address the question of whether such modified traffic characteristics have any impact on loss performance in the OBS network. In order to study the loss performance of the proposed and existing algorithms in an OBS network, we have chosen the topology given in Fig. 8 in which n access networks feed IP packets into a burst assembly buffer located at an OBS edge router which is connected to OBS core router using four wavelengths for data (bandwidth of each wavelength is set to 10 Gbps) and one wavelength for control. Packet arrivals from each access network is assumed to be Pareto on-off [1] with Hurst parameter $H = 0.8$, on-time $t_{\text{on}} = 5\,10^{-8}$, off-time $t_{\text{off}} = 5\,10^{-9}$ s with mean bit rate set to 0.8 Gbps. Packet size distribution is based on Table 1. We set $B_{\min} = 10$ kb and $B_{\max} = 70$ kb. The size of the burst header is assumed to be 125 bytes, the offset time is set to 40µs and simulation run-time is set to 20 s. When a burst assembly decision is to be made by the burst assembly unit and if all the wavelength channels are occupied after the offset time, this particular burst is assumed to be lost. We are interested in the probability of loss using various burst assembly methods. In Fig. 8, we increase the number of access networks (denoted by n) from 42 to 46 and we have set β to 3,000n. Under these conditions, we have compared the loss rates of various burst assemblers. Although the measured average burst size is about 35 kb for each assembly algorithm, we have observed that the *dyn-threshold-byte* algorithm significantly reduces the probability of loss in the bufferless core network as we see in Fig. 9. From this example, we conclude that the proposed algorithms not only reduce average packet or byte delays but the traffic they generate do not appear to have any adverse impact on the loss performance in the OBS network.

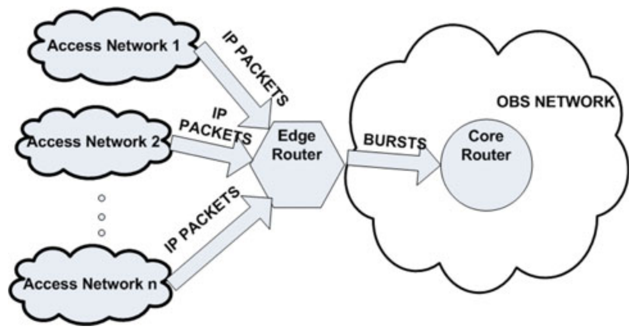

Fig. 8 Burst assembly scenario to study the probability of loss

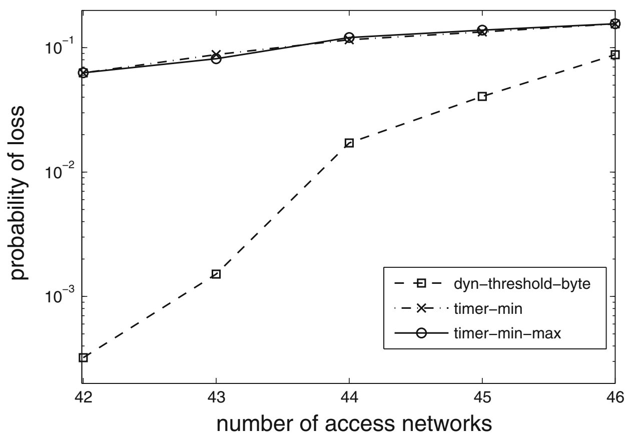

Fig. 9 Probability of loss as a function of the number of access networks n

5 Conclusions

In this study, we have proposed two dynamic-threshold based algorithms that aim at the reduction of average assembly delays (packet or byte delays) at burst assembly buffers located at the edge of an OBS network while conforming to a desired burst rate. Moreover, enforcement of lower and upper burst length limits is embedded in these algorithms. The major contribution of this article is the significant reduction of average assembly delays while keeping the short- and long-term burst rates close to the desired burst rate by means of dynamically adjusting the assembly threshold in case of changing traffic conditions. The benefits of the proposed algorithms are demonstrated with both synthetic traffic and actual traffic traces. Moreover, the algorithms are model-free and simple to implement making them viable alternatives for the design and implementation of burst assembly units in next-generation OBS systems.

Acknowledgements This work was supported in part by 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 the project EEEAG-106E046.

References

[1] Bohnert, T., Monteiro, E.: A comment on simulating LRD traffic with pareto ON/OFF sources. In: Proceedings of CoNEXT 2005, First International Conference on Emerging Networking Experiments and Technologies, pp. 228–229, Toulouse, France, Oct 2005

[2] Cao, X., Li, J., Chen, Y., Qiao, C.: Assembling TCP/IP packets in optical burst switched networks. In: Proceedings of IEEE GLOBECOM, Taipei, Taiwan, vol. 3, pp. 2808–2812, 17–21 Nov 2002

[3] Cho, K., Mitsuya, K., Kato, A.: Traffic data repository maintained by the MAWI Working Group of the WIDE Project. http://mawi.wide.ad.jp/mawi

[4] Du, P., Abe, S.: Burst assembly method with traffic shaping for the optical burst switching network. In: Proceedings of IEEE GLOBECOM, Session OPN-05, San Francisco, CA, USA, 27 Nov–1 Dec 2006

[5] Ge, A., Callegati, F., Tamil, L.: On optical burst switching and self-similar traffic. IEEE Commun. Lett. **4**(3), 98–100 (2000)

[6] Heffes, H., Lucantoni, D.: A Markov modulated characterization of packetized voice and date traffic and related statistical multiplexer performance. IEEE J. Select. Areas Commun. **4**(6), 856–868 (1986)

[7] Hong, J.H., Sohraby, K.: On the asymptotic analysis of packet aggregation systems. In: Proceedings of 15th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), pp. 353–359, Istanbul, Turkey, 24–26 Oct 2007

[8] Korkakakis, N., Vlachos, K.: An adaptive burst assembly scheme for OBS-GRID networks. In: Proceedings of 6th International Symposium on Communication Systems, Networks and Digital Signal Processing (CNSDSP), pp. 414–417, Graz, Austria, 25 July 2008

[9] Matisse Networks: http://www.matissenetworks.com/

[10] Muscariello, L., Meillia, M., Meo, M., Marsan, M., Cigno, R.: An MMPP-based hierarchical model of Internet traffic. In: Proceedings of IEEE International Conference on Communications (ICC), vol. 4, pp. 2143–2147, Paris, France, 20–24 June 2004

[11] Oh, S.Y., Hong, H.H., Kang, M.: A data burst assembly algorithm in optical burst switching networks. ETRI J. **24**(4), 311–322 (2002)

[12] Qiao, C., Yoo, M.: Optical burst switching (OBS)—a new paradigm for an optical Internet. J. High Speed Netw. **8**(1), 69–84 (1999)

[13] Sanghapi, J.N.T., Elbiaze, H., Zhani, M.: Adaptive burst assembly mechanism for OBS networks using control channel availability. In: Proceedings of 9th International Conference on Transparent Optical Networks (ICTON), vol. 3, pp. 96–100, Rome, Italy, 1–5 July 2007

[14] Verma, S., Chaskar, H., Ravikanth, R.: Optical burst switching: a viable solution for terabit IP backbone. IEEE Netw. Mag. **14**(6), 48–53 (2000)

[15] Vokkarane, V.M., Haridoss, K., Jue, J.P.: Threshold-based burst assembly policies for QoS support in optical burst-switched networks. In: Proceedings of SPIE OptiComm, pp. 125–136, Boston, MA, 29 July–2 Aug 2002

[16] Yu, X., Chen, Y., Qiao, C.: Study of traffic statistics of assembled burst traffic in optical burst switched networks. In: Proceedings of SPIE OptiComm, pp. 149–159, Boston, MA, 29 July–2 Aug 2002

Author Biographies

Mehmet Altan Toksöz received his B.S. degree in Electrical and Electronics Engineering from Anadolu University, Eskişehir, Turkey, in 2006 and M.S. degree in Electrical and Electronics Engineering from Bilkent University, Ankara, Turkey, in 2009. His current research interests are resource management algorithms for computer and communication networks.

Nail Akar received the B.S. degree from Middle East Technical University, Turkey, in 1987 and M.S. and Ph.D. degrees from Bilkent University, Turkey, in 1989 and 1994, respectively, all in Electrical and Electronics Engineering. From 1994 to 1996, he was a visiting scholar and a visiting assistant professor in the Computer Science Telecommunications program at the University of Missouri—Kansas City. He joined the Technology Planning and Integration group at Long Distance Division, Sprint, Overland Park, Kansas, in 1996, where he held a senior member of technical staff position from 1999 to 2000. Since 2000, he has been with Bilkent University, currently as an associate professor. His current research interests include performance analysis of computer and communication networks, optical networks, queueing systems, traffic control and resource allocation.