A token-based distributed algorithm to support QoS in a WDM ring network

Ashok Kumar Turuk, Rajeev Kumar [*]

Department of Computer Science and Engineering, Indian Institute of Technology, Kharagpur, WB 721302, India

Abstract

In this paper, we extend our recent work [Opt. Commun. 231(1–6) (2004) 199] by adding quality-of-service (QoS) provisioning. We revise the earlier proposed node architecture and the token-based distributed 'earliest available channel' algorithm to access the shared medium in a Wavelength Division Multiplexing ring network by supporting priority based QoS. The proposed algorithm which we call *earliest available channel with priority* (EACP) is based on a reservation scheme; however, it differs from the traditional reservation scheme in that no explicit release of reserved resources takes place. EACP algorithm gives priority to node having high priority request in reserving destination node and/or data-channel. The node architecture is configured around a tunable transceiver; thus the proposed scheme is scalable with respect to the number of wavelengths. We study the performance of the algorithm, by simulation, for bursty traffic modeled by M/Pareto distribution, and compare the performance with another token based algorithm. We found that our scheme performs superior in terms of wavelength utilization. However, delays are higher due to the single tunable transceiver in our scheme as opposed to an array of transmitters and receivers used in the other work.

Keywords: QoS; WDM ring network; Medium access control; Token; Tunable transceiver

1. Introduction

There has been a phenomenal growth in demand for bandwidth due to the ever increasing number of Internet users and increase in variety of Internet applications. It is widely believed that the next generation

[*] Corresponding author. Tel.: +91-3222283464; fax: +91-3222279579.
E-mail addresses: akturuk@cse.iitkgp.ernet.in (A.K. Turuk), rkumar@cse.iitkgp.ernet.in (R. Kumar).

doi:10.1016/j.optcom.2004.06.022

of optical Internet built on Wavelength Division Multiplexing (WDM) technology would satisfy the increasing demand for bandwidth. However, different Internet applications such as those involving multimedia traffic require different levels of quality-of-service (QoS). Today's Internet based on packet switching paradigms supports the best-effort service. WDM provides the required bandwidth, however, this does not guarantee the QoS requirements of different applications. Therefore, it is envisaged that the future optical Internet should not only meet the bandwidth requirements but also support the QoS requirements of different applications.

To provide end-to-end QoS the backbone networks as well as the Local Area Networks (LANs) must support some kind of QoS. The end-users of Internet applications are mostly hooked to a LAN. With increase in demand for bandwidth at the backbone network the corresponding demand at the LAN has increased proportionately. To meet the bandwidth requirements at LAN much work has been reported in the literature for the deployment of WDM technology in LANs. To provide end-to-end QoS not only the backbone network but also the LAN must support some kind of QoS. Available bandwidth is shared among all the network users in a LAN. To deal with multiuser access, a media access control (MAC) protocol is needed in such networks. In recent years, many media access control protocols have been proposed for WDM LAN based on star or ring as the underlying physical topology [2,3].

In this paper, we focus on a token-based WDM LAN with ring as the underlying physical topology. Token-based WDM ring is explained in Fumagalli et al. [4,5]. Unlike the *FDDI* rings, Fumagalli et al. [4,5] used multiple tokens in the ring. Number of tokens, number of transmitters and receivers at each node in the ring are equal to number of data-channels in their scheme [4,5]. Most of the MAC protocols proposed for WDM ring are based on the case where the nodes are equipped with either a single tunable transmitter and a fixed receiver (TT–FR), or a fixed transmitter and a fixed receiver (FT–FR). A few of them [5–8] require an array of transmitters and/or receivers at each node. Such architectures where nodes are equipped with fixed transmitters and/or receivers are not scalable. The major drawback of such nodes with an array of transmitters and receivers is that they give rise to high equipment cost and are not scalable. Additionally, when operating in a multi-traffic environment the MAC protocol should be able to interleave the different priorities of the traffic to meet their QoS requirements. To the best of our knowledge not much of the work is reported in the literature to provide QoS in a WDM ring network. For example, Marsan et al. [7] proposed an incremental slot reservation strategy based on local node traffic with *TT–FR* as the node architecture. Bengi et al. [8] proposed different access strategy for real-time and data traffic. Fumagalli et al. [4] proposed control channel based multi-token approach; the authors concluded that the approach can be extended to provide QoS. While the work reported in [7,8] used slotted ring, the work [4] used token-based ring. Number of data-channels is equal to the number of nodes in [7], while in [8] the number of nodes is greater than the number of channels. In [4], nodes are equipped with transceiver array imposing constraint on scalability.

The present work, in this paper, is an extension of our previous work describing EAC algorithm [1]. We revise the node architecture and the EAC algorithm to access the shared medium and to support QoS in an optical ring network. Like EAC, the revised algorithm which we call '*earliest available channel with priority*' (EACP) algorithm selects the earliest available data-channel. The protocol is based on a reservation scheme. To overcome the high reservation latency transmission of a node is overlapped with reservation. We used bursty traffic for simulating the performance of the *EACP* algorithm. We compare performance of the *EACP* algorithm with another token-based algorithm proposed by Fumagalli et al. [4]; their work is closest to our work.

The rest of the paper is organized as follows. In Section 2 the system model is described. In Section 3 proposed algorithm is presented. Correctness of algorithm is mentioned in Section 4. Simulation results are reported in Section 5. Finally, some conclusions are drawn in Section 6.

2. System model

2.1. Node architecture

A number of architectures for WDM LANs are reported in the literature; see [4,5,7–10] for ring topology, and see [11–20] for star topology. In this work, we propose a node architecture for token-based WDM ring network. Classically, a node architecture is equipped with a tunable transmitter and a fixed receiver, a fixed transmitter and a tunable receiver, or a fixed array of transmitters and receivers. Such architectures where nodes are equipped with fixed transmitter(s) and/or receiver(s) are not scalable. Moreover, in such a scheme, for any change in the spectrum requirements, the old fixed transmitters and/or receivers need to be replaced; this gives rise to high maintenance cost. To overcome the scalability problem of fixed transmitters and receivers, we propose an architecture where nodes are equipped with a single tunable transmitter for data-channels and a fixed transceiver for control channel; such a configuration is shown in Fig. 1. It is predicted by many that with the advances made in laser technology, the future nodes of WDM networks will be equipped with tunable lasers, of course, at a little additional cost. Tunable laser has several attractive features such as it can be remotely programmed to adjust the changing condition, and no replacement of laser is needed if the spectrum requirement changes [21].

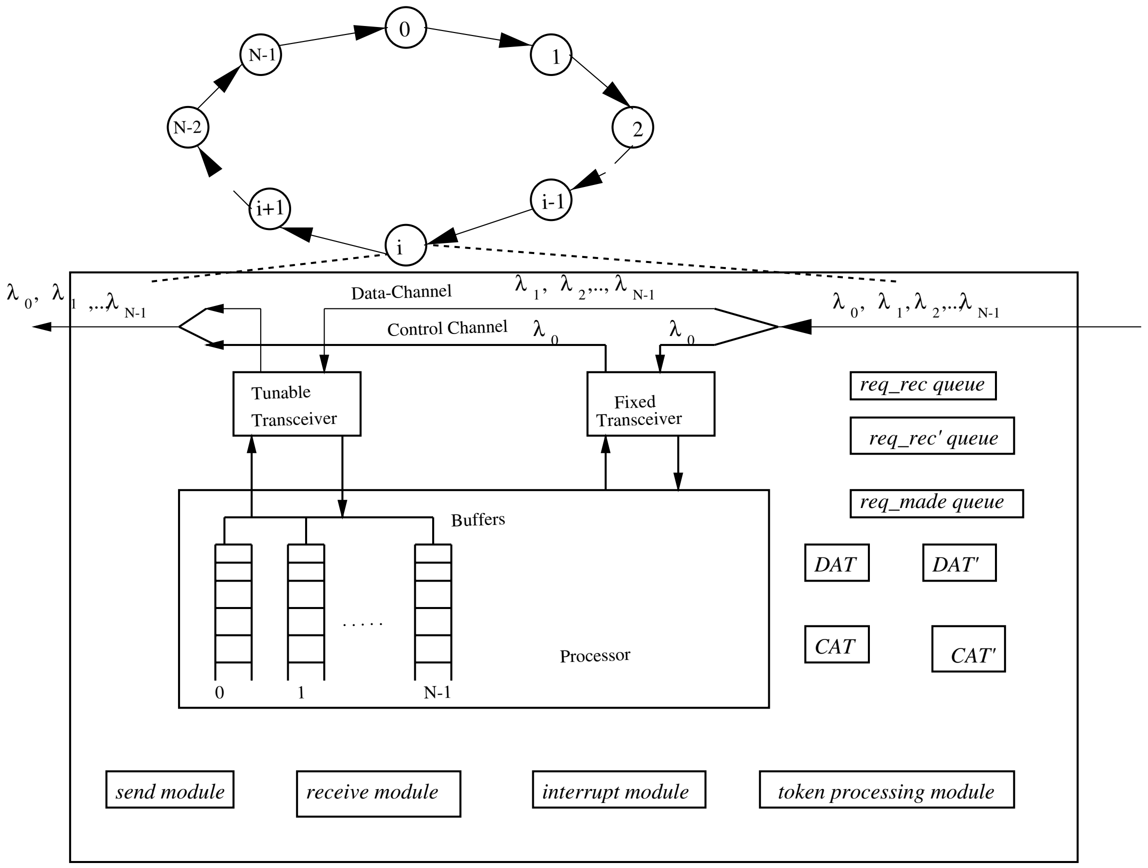

Fig. 1. Architecture of a node *i* in ring network.

The node architecture that we have proposed in this work is an extension of the node architecture used in our previous work [1]. In this current work, we propose a node having the following additional elements – a *receive module*, a *req_rec'* queue, and *DAT* and *CAT* vectors for supporting QoS. In the following paragraphs, from the completeness point of view, we include the complete details of the node architecture though this involves a bit of repetition.

We assume N nodes and W wavelengths in the network. One of the wavelengths is dedicated to control channel and the rest are used for data-channels. Each node is equipped with a single *fixed* transceiver for control channel and a *tunable* one for data-channels. There exists a fixed ADM for control channel and a tunable ADM for data-channel. The fixed transceiver is tuned to the wavelength dedicated for control channel to transmit and receive control information between adjacent nodes while the tunable transceiver is tuned to the data-channels as and when required. Each node in the network maintains status of its transmitter, receivers of the other nodes, and all the data-channels used in the network. Status gives the time at which transmitter, receivers, and data-channels are available. For two nodes to communicate, the tunable transmitter of the source and the tunable receiver of the destination must be tuned to the same wavelength (data-channel). Information transfer between the nodes takes a single hop over a data-channel.

In our proposed architecture, there is a *single* token that circulates around the ring on the control channel. The token consists of N fields, each field we call a *slot*; the *slot* l is assigned to node l. Each field is subdivided into *seven* mini-fields which we collectively call the *control information* of a slot. We define a *Token Period* (TP) as the period between two successive arrivals of the token at a node. We calculate TP as $TP = R + N \times p$, where p is the processing delay of token at each node and R is the associated ring latency. Since TP is same for all nodes in the network, each node gets a fair chance to access the shared medium. Thus, delay involved in accessing the token is bounded; this is a characteristic feature of a ring network.

A node on receiving the token processes each slot, l, $(0 \leqslant 1 < N)$ to update its knowledge about node l in the network. Prior to the communication between a pair of nodes, the source must reserve the destination and one data-channel. A node reserves the destination and the data-channel by writing the control information at its allotted slot in the token. We explain the reservation mechanism in Section 3. A node i writes control information in slot i of the token and modify slot j if the reservation request of node j is de-reserved at node i. Besides $N-1$ buffers, one for each destination, every node has *send*, *receive*, *interrupt*, *token processing* and *interrupt* modules; *req_rec*, *req_send* and *req_rec'* queues; *DAT*, CAT, *DAT'* and *CAT'* vectors; the purpose of each of these elements is explained in the following Section 2.2.

2.2. Notation and definitions

Each node i maintains the following four vectors: *DAT*, *CAT*, *DAT'* and *CAT'* vectors where:

$DAT[d]$: indicates the earliest time at which the receiver of node $d \neq i$ will be available for receiving;
$DAT[i]$: indicates the earliest time at which the transmitter of node i will be available for transmitting;
$CAT[c]$: indicates the earliest time at which the data-channel c will be available for transmission, and DAT' and CAT' are the copies of DAT and CAT vectors, respectively;
τ_t: transmitter available time
τ_d: receiver available time
τ_c: data-channel available time
t_u: tuning time of the transmitter/receiver
t_p: average propagation delay between source and destination
S_λ: a set of data-channels
S_i, S_i': a set of nodes whose request is de-reserved at node i
current_time: time at which an action is taken at the node.

The above includes the notations S_λ, S_i, S_i' which were not needed in EAC algorithm.

req_made queue: A FIFO queue that holds successful reservation requests made by a node. Each element of the queue has the following fields: *tt* – time at which transmitter of a node is tuned to a data-channel; *di* – identity of destination node to which transmission will take place; *dc* – wavelength to which the transmitter of a node will be tuned to; *td* – duration for which transmission will take place.

req_rec queue: A sorted queue that holds the reservation requests from other nodes for which the current node (here current node is the node that is processing the token) is the destination. For example, say, there is a reservation request from node 1, destined to node 5. When node 5 receives the token, reservation request from node 1 is added in its *req_rec* queue. No other node will make an entry of this request in it's *req_rec* queue. Elements of the *req_rec* queue are same as that of *req_made* queue. Here *dc* field specifies the wavelength to which the receiver of the node will be tuned to.

req_rec′ queue: A FIFO queue that holds the requests that are served by a node from its *req_rec* queue in the last *TP*.

T_x: indicates the status of a node's transmitter (FREE/BUSY), and
R_x: indicates the status of a node's receiver (FREE/BUSY).

related request: We define two requests from node i and j to be *related* if they are either requesting the same destination node $m \neq i,j$ or the same data-channel λ_k, and node i is requesting at t and node j at t' such that $t < t' < t + TP$.

Control information in a $slot_j(s,d,c,t_c,D,p,rm)$ of the token are:

s: value of *one* indicates node j is making request for reservation, and *zero* indicates either the reservation request made by the node j is de-reserved or node j is not making request for reservation. When s equal to *zero* the $slot_j$ will not be processed by a node;
d: identity of the destination node requested for reservation;
c: identity of the data-channel requested for reservation;
t_c: time at which the transmitter of the source and the receiver of the destination are tuned to data-channel c;
D: duration of transmission;
p: priority of reservation request; 0: low, 1: high;
rm: a vector of $N-1$ elements that specifies the low priority requests that are modified by node j having high priority request. Each element of the vector has the following sub-fields: *sn* – identity of the source node whose request was modified by node j; *dn* – identity of the destination node requested for reservation by node *sn*; *wc* – data-channel requested for reservation by node *sn*; *tt* – tuning time of the transmitter and receiver in the modified request. Initially all sub-fields are set to negative values.

We added two control fields p and rm in the architecture used for EACP in addition to the fields used for EAC algorithm,

3. EACP algorithm

EACP algorithm differs from the traditional reservation mechanism in the following aspects. First, no explicit release of reserved resources takes place. Second, unlike the traditional reservation mechanism where resources are reserved only when they are free, EACP algorithm looks ahead to find the time at which the required resources are available and reserves the resources from that point of time.

(The algorithm is an adaptation of scheduling-based earlier available time scheduling (EATS) algorithm [22] originally designed for WDM passive-star based LANs/MANs.)

EACP algorithm uses the *DAT* and *CAT* vectors to find the time at which the resources are available. Third, to overcome the high reservation latency, the transmission and reservation at a node is overlapped. Resources (source node transmitter, receiver of destination node, and a data-channel) are reserved for a duration which is determined at the time reservation request is made, and is different for different reservation requests. This assumption is natural in some applications such as file transfer or WWW down-loading. However, if the duration of transmission is unknown, this can be approximated. Since it takes a TP for a node to reserve the required resources depending on the arrival rate we can estimate the traffic for the TP and reserve the resources for this period. The reserved resources can be requested for reservation by another node after the requested period. This protocol does not necessitate the explicit release of reserved resources. Transmitter of the source and receiver of the destination are tuned to the same data-channel before communication between them takes place. In other words, a lightpath is dynamically established between the source and destination along the reserved data-channel for a period requested at the time of reservation. Establishing such a dynamic lightpath is made possible now with the availability of fast tuning lasers as reported in [23–26].

EACP algorithm gives priority to a node having high priority request in reserving a destination node and/or data-channel. Process of reservation begins when a node receives the token and completes when the node receives back the token. High priority requests are always successful in making reservation whereas a low priority request may or may not be successful. A low priority reservation request is un-successful when a node having high priority request de-reserves it. A low priority request is de-reserved at a node by setting the *s* field of the *slot* corresponding to that request to *zero*. De-reserved requests are not processed at successive nodes. To avoid starvation of low priority request, the request priority is upgraded to high after a failure of certain number of reservation attempts. Request whose priority is changed to high is treated as a new request to the node, so that the previously arrived high priority requests are served first. Period of transmission is determined at the time reservation request is made. Duration of transmission for which a request is made is determined at each node by monitoring the traffic at the node for the last *TP* period.

EACP algorithm has the following modules: *Send*, *Receive*, *Token processing*, and *Interrupt* module. *send* and *receive* modules remain the same as that of EAC algorithm [1], however, EAC and EACP algorithms differ in the token processing module. Therefore, in the following paragraphs, we detail the token processing and interrupt modules; see [1] for details of *send* and *receive* modules.

Interrupt module: a reservation request destined to a node is added in the *req_rec* queue of that node. A low priority request that is added to *req_rec* queue may or may not be successful in making reservation. Such low priority requests that are not successful in making reservation and are added to the *req_rec* queue of the destination node must be removed from the *req_rec* queue or appropriate action must be taken such that the un-successful request is not processed. Later we show that a low priority request that is unsuccessful in making reservation and is added to the *req_rec* queue of the destination node is either removed from the *req_rec* queue or is not processed by the *receive* module. *Interrupt* module is invoked to terminate the processing of a unsuccessful request by the *receive* module.

Token processing module: when a node receives the token it invokes its *token processing* module and the following actions are taken. For a successful request made by the node: (i) value of the *s* field of its own slot is set to *zero*; (ii) available time of it's transmitter, destination node's receiver and reserved data-channel is updated; (iii) the successful request is added to its *req_made* queue; (iv) the *rm* vector of it's own slot is set to a negative value. Next, all high priority requests are checked to find if any of the high priority request has modified a low priority request. If such a request exists, the values of *DAT* and *CAT* vectors are restored back to their previous values. Before updating the values of *DAT* and *CAT* vectors a copy of it is stored in DAT' and CAT' vectors, respectively, so that it can be restored back to its previous value if required. If the modified low priority request exists in the *req_rec* queue of the node it is deleted from the queue. If the *receive* module has removed the request from the *req_rec* queue the *Interrupt* module is invoked to stop processing the request. Values of the DAT and CAT vectors are updated after storing a copy in DAT' and CAT' vectors, respectively.

Then the node makes reservation requests for which following actions are taken. If the buffers are non-empty, a high priority burst is selected if it exists otherwise a low priority burst is selected. Destination identity of the burst which has the maximum waiting time is found, and an earliest available data-channel is selected. The maximum of the available time among the node's transmitter, destination node's receiver and the selected data-channel is found. Let this time be t' and this gives the time at which all the required resources – source node's transmitter, destination node's receiver, and the selected data-channel – are available at the same time. The node can reserve the resources at t'. Let t be the time at which the node has received the token and it's reservation process is completed at $t+TP$. Then, $t'<t+TP$ implies the required resources are available before the reservation is completed. But a node can reserve the required resources only after it's reservation request is completed i.e., on or after $t+TP$ period of time. Therefore, if $t'<t+TP$ the value of t' is set to $t+TP$. Control information is written at the slot allotted to the node in the token and the token is sent to the successor node. When the destination node receives the token, it adds the request in its *req_rec* queue. When the source receives back the token its reservation process is completed and if the request is successful it is added in its *req_made* queue.

In the following subsection, we write the pseudocode for the EACP algorithm.

3.1. Pseudocode for the EACP – algorithm

Perform the following *Cases* at each node *i*; in the following pseudocode, node *i* is the node that is processing the module.

```
CASE 1: if (req_made queue is non-empty and $T_x = FREE$) invoke the Send module.
CASE 2: if (req_rec queue is non-empty and $R_x = FREE$) invoke the Receive module.
CASE 3: invoke the Token processing module when node i receives the token.
```

3.1.1. Interrupt module

```
1. Stop processing the request.
2. $R_x \leftarrow FREE$.
```

3.1.2. Token processing module

```
1. Examine $slot_i(s,d,c,t_c,D,p,rm)$ of the token. if ($s=1$) then do the following
    • $slot_i(s \leftarrow 0,\ rm \leftarrow a\ negative\ value)$
    • Add reservation request in req_send queue of node i.
    • $DAT[d] \leftarrow CAT[c] \leftarrow DAT[i] \leftarrow t_c + t_u + t_p + D$
2. for all $slot_{j \neq i}(s,d,c,t_c,D,p,rm)$ of the token, if ($s=1$ and $p=1$) do the following
    • while ($rm[k].sn \geqslant 0$){
    – switch($rm[k].sn \geqslant 0$){
        * case 0 or $N-1$:
        **if** ($i \geqslant mod(rm[k].sn+1,N)$ and ($i \leqslant \mathrm{mod}(j-1,N)$)){
          · $DAT[rm[k].dn] \leftarrow DAT'[rm[k].dn]$
          · $CAT[rm[k].wc] \leftarrow CAT'[rm[k].wc]$
          · **if** ($rm[k].dn = i$) delete the reservation request *if* exists in the *req_rec* queue node
            *i else* invoke interrupt module
          · break
        } // end of **if**, and also end of **Case** 0 or N-1
        * default:
        **if** ($i \geqslant mod(rm[k].sn+1,N)$ or ($i \leqslant mod(j-1,N)$)){
          · $DAT[rm[k].dn] \leftarrow DAT'[rm[k].dn]$
          · $CAT[rm[k].wc] \leftarrow CAT'[rm[k].wc]$
          · **if** ($rm[k].dn = i$) delete the reservation request *if* exists in the *req_rec* queue node
            *i else* invoke interrupt module
          · break
        } // end of **if**, and also end of **Case** default
        } // end of switch
      } // end of while
      end_for.
   3. $DAT' \leftarrow DAT$, and $CAT' \leftarrow CAT$
   4. *for* all $slot_{j \neq i}(s,d,c,t_c,D,p,rm)$ of the token, **if** ($s=1$) do the following
      • **if** ($t_c + t_u + t_p + D > DAT[d]$)
        – $DAT[d] \leftarrow CAT[c] \leftarrow t_u + t_c + t_p + D$
      • **if** ($p=1$)
        – $DAT'[d] \leftarrow DAT[d]$
        – $CAT'[c] \leftarrow CAT[c]$
      end_for.
   5. *if* node *i*'s buffer is empty goto Step 27.
   6. *if* node *i* has no high priority bursts to transmit goto Step 20.
   7. Find a burst with maximum waiting time. Let the destination identity of the burst be say $x$.
   8. *for* all $slot_{j \neq i,x}(s=1,\ d=x,\ p=o)$ do the following
      • add the wavelength requested by node $j$ to $S_\lambda$
      • de-reserve the request of node $j$
      • add node $j$ to $S_i$
      • record the destination node, data-channel and tuning_time of node $j$'s reservation request in *rm* vector
        of *slot i* .
      • $DAT[x] \leftarrow DAT'[x]$, $CAT[c] \leftarrow CAT'[c]$
      end_for.
   9. **if** ($S_\lambda \neq \phi$) do the following
      • $DAT[x] \leftarrow DAT'[x]$;
```

```
    • $CAT[\lambda_m] \leftarrow CAT'[\lambda_m]$ for all $\lambda_m \in S_\lambda$

10. Find the earliest available data-channel $k \leftarrow \{m$: $CAT'[m]$ is minimum for $m \leftarrow 1,\ldots,W-1\}$.
11. *for* all $slot_j(s=1,\ d\neq x,\ c=k,\ p=0)$ do the following
     • add node $j$ to $S_i'$
     • de-reserve the request of node $j$
     • record the destination node, data-channel and tuning_time of node $j$'s reservation
       request in $rm$ vector of *slot i*.
     • $DAT[x] \leftarrow DAT'[x]$, $CAT[c] \leftarrow CAT'[c]$
    end_for.
12. **while** $(S_i \neq \phi)$ do the following
     • remove a node $j$ from $S_i$
     • *if* the request of node $j$ is *related* with another request from node $n$ do the following
        – de-reserve the request of node $n$
        – add node $n$ to $S_i'$
        – record the destination node, data-channel and tuning time of node $n$'s reservation
          request in $rm$ vector of *slot i*.
        – $DAT[x] \leftarrow DAT'[x]$, $CAT[c] \leftarrow CAT'[c]$
13. **while** $(S_i' \neq \phi)$ do the following
     • remove an element $j$ from $S_i'$
     • if the request of node $i$ is *related* with another request from node $n$ do the following
        – de-reserve the request of node $n$
        – add node $n$ to $S_i$
        – record the destination node, data-channel and tuning time of node $n$'s reservation
          request in $rm$ vector of *slot i*.
        – $DAT[x] \leftarrow DAT'[x]$, $CAT[c] \leftarrow CAT'[c]$
14. **if** $(S_i \neq \phi)$ goto Step 12.
15. $\tau_t \leftarrow DAT[i]$, $\tau_d \leftarrow DAT[x]$, $\tau_c \leftarrow CAT[k]$
16. $\tau \leftarrow max(\tau_t,\tau_d,\tau_c)$. This gives the earliest time at which transmitter of source node $i$, receiver of
    destination node $x$ and a data-channel $k$ are available at the same time.
17. **if** $(\tau < current\_time + TP)$ $\tau = current\_time + TP$
18. Calculate the duration of transmission $D$.
19. Prepare $slot_i(s \leftarrow 1,\ d \leftarrow x,\ c \leftarrow k,\ t_c \leftarrow \tau,\ D,\ p \leftarrow 1,\ rm)$ goto Step 27.
20. Find a burst with maximum waiting time. Let the destination identity of the burst be say $x$.
21. Find the earliest available data-channel $k \leftarrow \{m$: $CAT[m]$ is minimum for $m \leftarrow 1,2,\ldots,W-1\}$.
22. $\tau_t \leftarrow DAT[i]$, $\tau_d \leftarrow DAT[x]$, $\tau_c \leftarrow CAT[k]$
23. $\tau \leftarrow max(\tau_d,\tau_d,\tau_c)$
24. **if** $(\tau < current\_time + TP)$ $\tau = current\_time + TP$
25. Calculate the transmission duration, $D$
26. Prepare $slot_i(s \leftarrow 1,\ d \leftarrow x,\ c \leftarrow k,\ t_c \leftarrow \tau,\ D,\ p \leftarrow 0,\ rm)$
```

```
27. for all $slot_{j \neq i}(s,d,c,t_c,D,p,rm)$ of the token, if ($s=1$ and $d=i$) do the following
      • Add the request of node $j$ in req_rec queue of node $i$.
    end_for.
28. Send the token to the successor node.
```

Send and *receive* modules are identical to those given for EAC algorithm so we have not reproduced here.

3.1.3. Simulation of EACP algorithm

We illustrate the reservation process in EACP algorithm with the following example. For simplification, we consider a four-node ring network and two number of data-channels. Available time of the transmitter and the receiver of each node, and data-channels is shown in Table 1. Table 2 shows the traffic at each node at some point of time t. The entry $b_i(x,y)$ corresponding to row m and column n of Table 2 indicates, source m has a burst of ith priority (*zero* for low and *one* for high) to transmit to destination n. Duration of transmission of the burst is indicated by x, and y indicates the time at which the burst has arrived at node m destined to node n. We choose the following quanta of values for our example: $t_u=2$, propagation delay of token between a pair of adjacent node to be 5, and the processing delay of token at each node is assumed to be negligible. Computed value of $TP=20$, $t_p=10$ (computed as in [27]).

Let $t=40$ and node 0 receives the token at t. Suppose the token has no reservation request. Contents of DAT and CAT vectors at node 0 remain unchanged after processing the token. Node 0 selects the destination node 2 and data-channel, λ_1. Following computation is performed: $\tau_t=DAT[0]$ i.e., 45, $\tau_d=DAT[2]$ i.e., 45, $\tau_c=CAT[\lambda_1]$ i.e., 47, $\tau=\max(\tau_t,\tau_d,\tau_c)$ i.e., 47. $\tau<t+t_u+TP$ so the value of τ is set to $t+TP$ i.e., 60. Control information is written in $slot_0(s=1, d=2, c=\lambda_1, t_c=\tau, D=4, p=0, rm)$ of the token and the token is send to node 1.

Node 1 updates the values of DAT and CAT vectors at its node shown in Table 4. Before updating, a copy of DAT and CAT vectors is stored in DAT' and CAT' vectors, respectively, as shown in Table 3.

Table 1
Available time of transmitter and receiver of nodes and data-channels

Node	0	1	2	3
Transmitter	45	40	110	47
Receiver	40	47	45	110
$CAT[\lambda_1]=47$		$CAT[\lambda_2]=110$		

Table 2
Traffic at different nodes

Node	0	1	2	3
0		$b_0(4, 23)$		
1	$b_0(10, 23)$			
2		$b_1(25, 33)$		
3	$b_1(10, 25)$	$b_0(25, 35)$		

Table 3
Contents of *DAT′* and *CAT′* vectors at node 1

$DAT'[0]=40$, $DAT'[1]=40$, $DAT'[2]=45$, $DAT'[3]=110$
$CAT'[\lambda_1]=47$, $CAT'[\lambda_2]=110$

Table 4
Contents of *DAT* and *CAT* vectors at node 1

DAT[0] = 40, DAT[1] = 40, DAT[2] = 76, DAT[3] = 110
CAT[λ_1] = 76, CAT[λ_2] = 110

Node 1 selects the destination node 0 and data-channel λ_1, and performs the following computation as stated earlier: $\tau_t=40$, $\tau_d=40$, $\tau_c=76$, $\tau=76$. Control information is written in $slot_1(s=1$, $d=0$, $c=\lambda_1$, $t_c=\tau$, $D=10$, $p=0$, $rm)$ of the token and the token is send to node 2.

Node 2 selects the destination node 1 and data-channel λ_1. The requests from node 0 and node 1 are de-reserved at node 2, and are recorded in the *rm* vector of $slot_2$ of the token (requests from node 0 and node 1 are of low priority and are requesting data-channel λ_1 which is selected by node 2 having high priority request). The values of *DAT* and *CAT* vectors after processing the token are shown in Table 5. Following computation is performed: $\tau_t=110$, $\tau_d=47$, $\tau_c=47$, $\tau=110$. Control information is written in $slot_2(s=1$, $d=1$, $c=\lambda_1$, $t_c=\tau$, $D=25$, $p=1$, $rm)$ of the token and the token is sent to node 3.

The values of *DAT* and *CAT* vectors at node 3 after processing the token are shown in Table 6.

Node 3 selects destination node 0 and data-channel λ_2. Following computation is performed: $\tau_t=47$, $\tau_d=40$, $\tau_c=110$, $\tau=110$. Control information in $slot_3(s=1$, $d=0$, $c=\lambda_2$, $t_c=\tau$, $D=10$, $p=1$, $rm)$ of the token and the token is sent to node 0.

When node 0 receives the token, it finds its reservation request is not successful, and makes another reservation attempt. The values of *DAT* and *CAT* vectors after processing the token are shown in Table 7.

Table 5
Contents of *DAT* and *CAT* vectors at node 2

DAT[0] = 40, DAT[1] = 47, DAT[2] = 110, DAT[3] = 110
CAT[λ_1] = 47, CAT[λ_2] = 110

Table 6
Contents of *DAT* and *CAT* vectors at node 3

DAT[0] = 40, DAT[1] = 147, DAT[2] = 45, DAT[3] = 47
CAT[λ_1] = 147, CAT[λ_2] = 110

Table 7
Contents of *DAT* and *CAT* vectors at node 0

DAT[0] = 45, DAT[1] = 147, DAT[2] = 45, DAT[3] = 110
CAT[λ_1] = 147, CAT[λ_2] = 132

Table 8
Contents of *DAT* and *CAT* vectors at node 1

DAT[0] = 132, DAT[1] = 40, DAT[2] = 148, DAT[3] = 110
CAT[λ_1] = 147, CAT[λ_2] = 148

Request from node 3 is entered in the *req_rec* of node 0. Node 0 selects the destination node 2 and data-channel λ_2, and performs the request operation as explained earlier.

When node 1 receives the token the values of $DAT[2]$ and $CAT[\lambda_1]$ are restored to the previous values by setting $DAT[2] = DAT'[2]$, and $CAT[\lambda_1] = CAT'[\lambda_1]$. Note that the request of node 0 was de-reserved at node 2 and node 1 has updated the parameters corresponding to this request which needs to be restored back before processing further for correct operation of the algorithm. The values of *DAT* and *CAT* vectors at node 1 after processing the token are shown in Table 8. Request from node 2 is entered in the *req_rec* queue of node 1.

Reservation process continues as explained above. Note before a node updates values of its *DAT* and *CAT* vectors the copies are stored in DAT' and CAT' vectors, respectively, so that these can be restored back to the previous values if the request become un-successful. In the above example, we have shown for updating values of DAT' and CAT' vectors for node 1 only; they need to be updated for every node.

4. Correctness of the algorithm

In the following subsections, we illustrate correctness of the algorithms that: (i) Destination collision never occurs, (ii) Channel collision never occurs, (iii) Transmitter of the source and Receiver of the destination are tuned to the same data-channel precisely at the same time, (iv) Reservation requests made by a node do not overlap in time, (v) Reservation requests received by a node do not overlap in time, (vi) Low priority requests that are added in the *req_rec* queue of the destination node and are unsuccessful in making reservation are either removed from the *req_rec* queue or are not processed by the *receive* module. (vii) If a low priority request is de-reserved at a node having high priority request, then the parameters that are updated at nodes corresponding to the de-reserved request are correctly restored back to its previous value.

The correctness illustrations for the cases (i)–(v) remain the same as given in [1]. The remaining illustrations for cases (vi) and (vii) are given below.

4.1. Low priority requests that are added in the req_rec queue of the destination node and are unsuccessful in making reservation are either removed from the req_rec queue or not processed by the receive module

Let node x make low priority reservation request destined to node y, which is de-reserved by node z having high priority request. Suppose node y has entered the request from node x in its *req_rec* queue before node z de-reserves the request. We show that the request of node x is either not processed, or removed from the *req_rec* queue of node y. We consider two cases:

Case 1. The *receive* module of node y has removed the request of node x from the *req_rec* queue node y.

We know that when a request is removed from the *req_rec* queue of a node, a copy of it is added to the *req_rec′* queue of that node. This is done in step 1 of the *receive* module [1]. When node y receives back the token, the *req_rec′* queue is checked for the de-reserved request of node x. If the de-reserved request of node x is at the end of the *req_rec′* queue, it indicates the *receive* module of node y has most recently removed the request of node x from the *req_rec* queue of node y. So, the *Interrupt processing* module is called, which stops processing the request from node x and sets the receiver of node y to *FREE*. Thus the request from node x is not processed further.

Case 2. The *receive* module of node y has not removed the request of node x from the *req_rec* queue node y.

When node y receives back the token, the de-reserved request of node x is deleted from the *req_rec* queue of node y. This is done in step 2 of the *token processing* module. Hence, the de-reserve request is not available for processing.

4.2. If a low priority request is de-reserved at a node having high priority request, then the parameters that are updated at nodes corresponding to the de-reserved request are correctly restored back to previous values

We consider different cases to show that the parameters updated by de-reserved requests are restored back to their previous values (the values before updating), giving correct operation of the algorithms.

Case 1. Suppose nodes s_1 and s_2 have requests destined to node d, and priority of requests be low and high, respectively. Let node s_1 receive the token at t and make reservation request. λ_1 be the selected data-channel. The intermediate nodes between s_1 and s_2 (both exclusive) update the values of $DAT[d]$ and $CAT[\lambda_1]$ at their nodes when they receive the token, mentioned in step 4 of the *token processing* module. A node makes a copy of the values of DAT and CAT at it's node in DAT' and CAT', respectively, before updating the values of DAT and CAT. This is done in step 3 of the *token processing* module.

Let node s_2 receive the token at t' where $t < t' < t + TP$. Node s_2 has high priority request destined to node d. Request from node s_1 is de-reserved at node s_2. Note both the algorithms give priority to node having high priority requests in reserving the destination node. Node s_2 records the de-reserve request from node s_1 in its control information field that is in the *rm* vector of $slot_2$ of the token. This is done in step 8 of the *token processing* module. A de-reserve request is not processed by other nodes. When the intermediate nodes between s_1 and s_2 receive back the token the values of $DAT[d]$ and $CAT[\lambda_1]$ are restored back to their previous values. This is done is step 1 of *token processing* module.

Thus the values of DAT and CAT vectors updated at nodes due to de-reserved low priority requests are restored back to its previous values.

Case 2. Suppose nodes s_1 and s_2 have requests destined to node d_1 and d_2, and priority of requests be low and high, respectively. Let node s_1 receive the token at t and make reservation request. λ be the selected data-channel. The intermediate nodes between s_1 and s_2 (both exclusive) update the values of $DAT[d_1]$ and $CAT[\lambda]$ at their nodes when they receive the token, this is done in step 4 of the *token processing* module. As already stated a node makes a copy of the values of DAT and CAT at its node in DAT' and CAT', respectively, before it updates DAT and CAT vectors.

Let node s_2 receive the token at t' where $t < t' < t + TP$. Suppose the data-channel selected at node s_2 be λ which is also selected by node s_1 having low priority request. Request from node s_1 is de-reserved at node s_2. Note both of the algorithms give priority to node having high priority request in reserving data-channel. Node s_2 records the de-reserve request from node s_1 in its control information field. This is done in step 11 of the *token processing* module.

When the intermediate nodes between s_1 and s_2 receive back the token the values of $DAT[d_1]$ and $CAT[\lambda]$ are restored back to their previous values. This is done is step 1 of *token processing* module.

Thus the values of DAT and CAT updated at nodes due to de-reserved low priority requests are restored back to previous values.

Case 3. Suppose the requests of nodes i and j are related. If the request of node i is de-reserved by a node z having high priority request, then the request of node j must be de-reserved. This is because the transmission from node j follows the completion of transmission of node i, and the duration of transmission of node i may be different from that of node z.

Suppose nodes s_1 and s_2 have low priority requests destined to node d_1 and the data-channel selected for reservation be λ_1 and λ_2, respectively. Suppose the requests of node s_1 and s_2 are also related. Suppose node s_3 has high priority request to node d_3 and the data-channels selected be λ_1. When node s_3 receives the token the values of $DAT[d_1]$, $CAT[\lambda_1]$ and $CAT[\lambda_2]$ have updated appropriately at the intermediate nodes. Node s_3 has higher priority request, and selected data-channel λ_1 which is also selected by node s_1 having low priority request. So, the request of s_1 is de-reserved at node s_3, this is done in step 11 of the *token processing* module. Requests from nodes s_1 and s_2 are related by assumption and the request from node s_1 is de-reserved at node s_3. This results in the de-reservation of requests from node s_2 at node s_3. This is done in step 14 of the *token processing* module.

The steps 12–14 of the *token processing* module de-reserve all *related* requests. All de-reserved requests are recorded in the *rm* vector of the control information field of the node that de-reserves the request. In the case that we considered above, de-reserve requests are recorded in the *rm* vector of control information field of node s_3.

Updated values of DAT and CAT vectors restored back as explained in case 1.

Case 4. Suppose nodes s_1 and s_2 have low priority requests destined to nodes d_1 and d_2, respectively, and λ be the selected data-channel. Suppose the requests from nodes s_1 and s_2 are also *related*. Let node s_3 has high priority request destined to node d_1. The low priority request from node s_1 is de-reserved at node s_3 (requests of both s_3 and s_1 are destined to node d_1 and d_3 having higher priority). This is done in step 3 of the *token processing* module. Requests from node s_1 and s_2 are *related*, this results in the de-reservation of request from node s_2. Steps 14–16 de-reserve all *related* requests. All de-reserved requests are recorded in the *rm* vector of the control information field of the node that de-reserves the requests. In the case that we considered above, de-reserve requests are recorded in the *rm* vector of control information field of node s_3. Updated values of DAT and CAT vectors are restored back as explained in case 1.

The above illustrates correctness of the EACP algorithm.

5. Simulation and results

In this section, we evaluate the performance of the proposed node architecture and EACP algorithm by simulation. We measure the performance in terms of wavelength utilization, throughput (bps) and mean delay; we use these three matrics as they are first class design parameters in WDM networks and used by many authors. We consider a 10 node ring network where nodes are equally spaced around the ring. The number of data-channels are varied between 5 and 7. We use the following quanta of values in carrying-out the simulation: capacity of data-channel is 1 Gbps, length of the ring is 100 km (*FDDI* ring can span upto 200 km [28]), processing time of token at each node is 1 μs, and tuning time of transmitter and receiver is 5 μs (laser with tuning time of 5 ns are reported in the literature see [12–15]). Computed value of TP comes out to be 510 μs from the formulation given in Section 2.1. The average propagation delay, t_p, between nodes is computed by $t_p = N/2 \times \tau$ where τ is the propagation delay between adjacent nodes [29]. Further, we assume that a lightpath is established between source and destination during a TP. We consider bursty traffic as the traffic in LANs are reported to be bursty in nature [30]. We use M/Pareto distribution for generating the bursts [31]. We keep the size of packets in a burst to be fixed at 10 kb per packet. We assume that the load in the network is uniformly balanced.

For simulation, we have considered the following three cases:

- Case 1: both size and inter-arrivals of bursts are deterministic which we call *DaDb*.
- Case 2: inter-arrival of bursts to follow an exponential distribution and deterministic burst size which we call *EaDb*.

- Case 3: inter-arrival of bursts to follow an exponential distribution and burst size follows M/Pareto distribution we call *EaPb*.

We define the terms *DaDb*, *EaDb* and *EaPb* using the following notations:

D: deterministic size,
E: exponential distribution,
P: M/Pareto distribution,
a: inter-arrival of burst,
b: burst size.

In the above, the *D*, *P* and *E* are suffixed by *a* and *b*, which we use for denoting inter-arrival of the bursts and burst size, respectively. For example, *EaPb* implies exponential inter-arrival of bursts and Pareto distributed burst-size. For deterministic cases, the size of the burst and the inter-arrival of the bursts are known. We have considered the deterministic cases for comparison purpose only. However, in the real-world scenario neither the burst size nor the inter-arrival of burst are known in advance. Therefore, the real-world scenario is better modeled by *EaPb*. Most of the results presented in the rest of this section belong to this case.

This section is divided in two parts. In first Section 5.1, we include the simulation results obtained from the EACP algorithm. Then, in next Section 5.2 we compare the performance of EACP algorithm with the available algorithm closest to our work and proposed by Fumagalli et al. [4].

5.1. Simulation results of EACP algorithm

5.1.1. Wavelength utilization vs. burst size

First, we include plots for wavelength utilization vs. burst size – Figs. 2–4. Burst size is expressed in number of packets. We generated one million packets in our simulation for each study. Wavelength utilization for fixed-size burst is plotted in Fig. 2. The inter-arrival of bursts is assumed to be 1 ms. The wavelength utilization increases with increase in the burst size and at a higher burst size the wavelength utilization saturates. The increase in wavelength utilization in the lower range of burst-sizes is mainly due to the increase in the transmission duration. For example, at a burst size of 50 packets the duration of transmission is 500 μs whereas at a burst size of 200 packets the duration of transmission is 1 ms. At a higher burst size the duration of transmission increases but the scheduling latency also increases proportionately giving

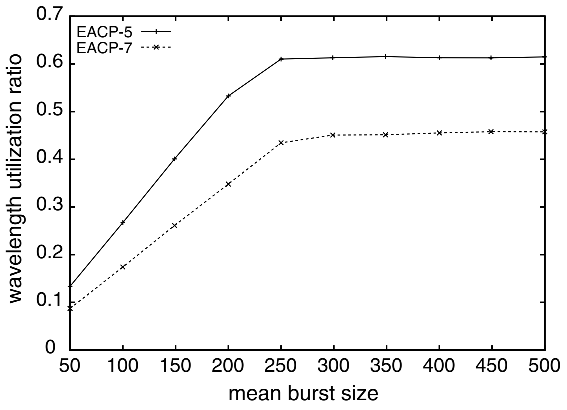

Fig. 2. Burst size vs. wavelength utilization for *FaFb*.

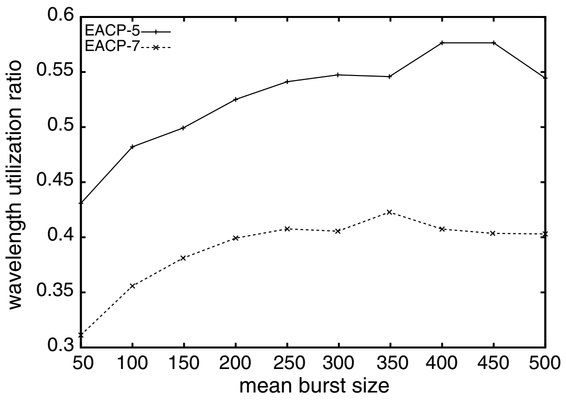

Fig. 3. Burst size vs. wavelength utilization for *EaFb*.

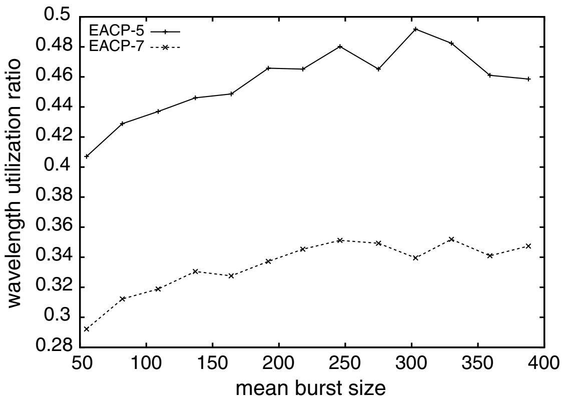

Fig. 4. Burst size vs. wavelength utilization for *EaPb*.

an almost constant utilization in wavelength. (We have taken scheduling latency as the difference in time between the start of the transmission on the channel and the availability of the channel.)

Next, we include results of wavelength utilization for exponential inter-arrival and fixed-sized bursts in Fig. 3. It is observed the wavelength utilization increases marginally with increase in the burst size. The marginal increase in the wavelength utilization is due to the proportionate increases in the scheduling latency with the increase in the duration of transmission. Similar is the trend for exponential inter-arrival and Pareto distributed burst size as shown in the Fig. 4.

From Figs. 2–4, it is also observed that the wavelength utilization decreases with increase in the number of wavelengths. This is due to the fixed number of packets (which is one million in our present case) that we have generated in our simulation. These packets are transmitted over the increased wavelength giving lesser wavelength utilization.

5.1.2. Throughput vs. burst size

The plots for throughput vs. burst size are included in Figs. 5–7. Throughput for fixed inter-arrival time and burst size is plotted in Fig. 5. It is observed that throughput increases for lower burst size and gets sat-

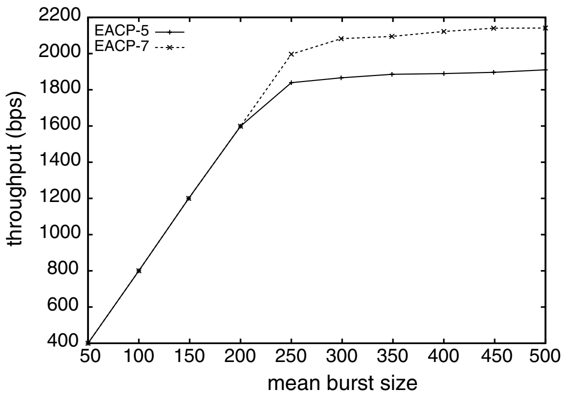

Fig. 5. Burst size vs. throughput (bps) for *FaFb*.

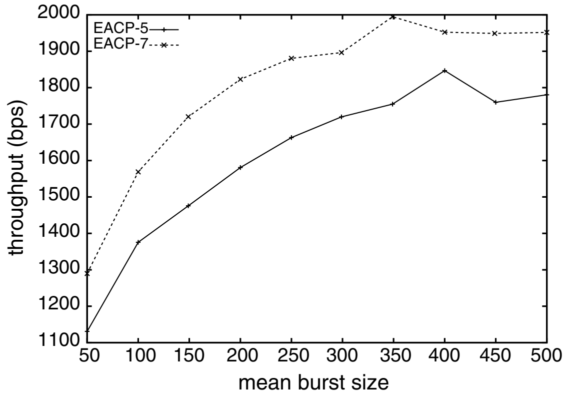

Fig. 6. Burst size vs. throughput (bps) for *EaFb*.

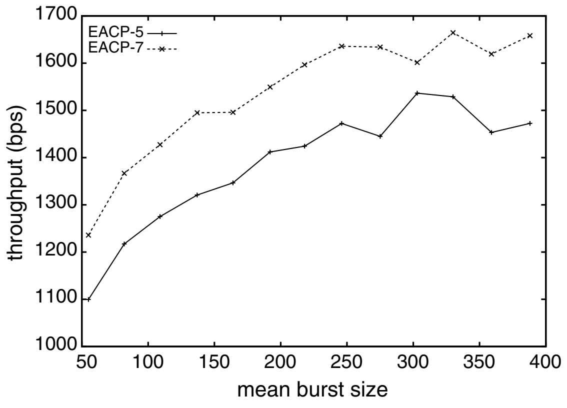

Fig. 7. Burst size vs. throughput (bps) for *EaPb*.

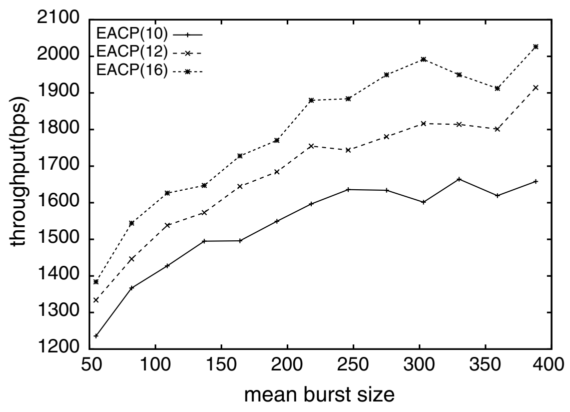

Fig. 8. Burst size vs. throughput (bps) for *EaPb* with 10, 12 and 16 nodes and seven wavelengths (the number inscribed in parenthesis in the graph represents the number of nodes used).

urated at higher burst size; this is because of the increase in the wavelength utilization at lower burst size and gets saturated at higher burst size as shown in Fig. 2.

Throughput for exponential inter-arrival time of burst and fixed burst size is plotted in Fig. 6. It is observed that throughput increases with increase in the burst size. Almost similar is the trend for exponential inter-arrival time and Pareto distributed burst size as shown in Fig. 7. The increase in the throughput is due to the increase in the wavelength utilization as shown in Figs. 3 and 4, respectively.

Fig. 8 shows the throughput for different number of nodes at a fixed number of wavelengths. We varied the number of nodes, and used 10, 12 and 16 node-topologies, and kept the number of wavelengths fixed at seven. It is observed that with increase in the number of nodes the throughput increases. This is because with increase in the number of nodes, more nodes have data to transmit giving better wavelength utilization and higher throughput.

5.1.3. *Burst size vs. mean delay*

Plots for mean delay vs. burst size are included in Figs. 9–11. Mean delay for fixed arrival and burst size is plotted in Fig. 9. It is observed that at lower burst size delay experienced by packets is lesser and the delay

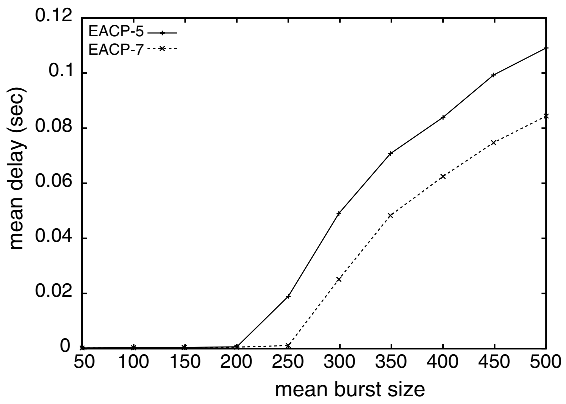

Fig. 9. Burst size vs. mean delay for *FaFb*.

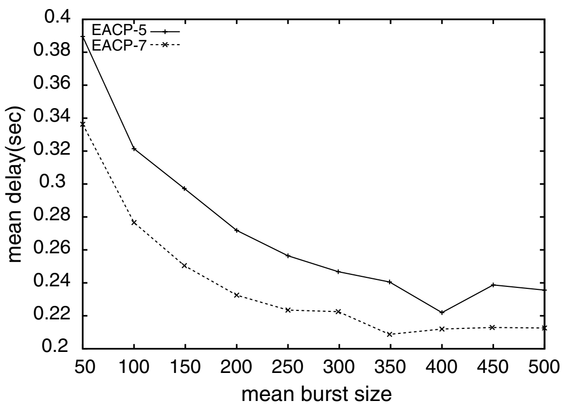

Fig. 10. Burst size vs. mean delay for *EaFb*.

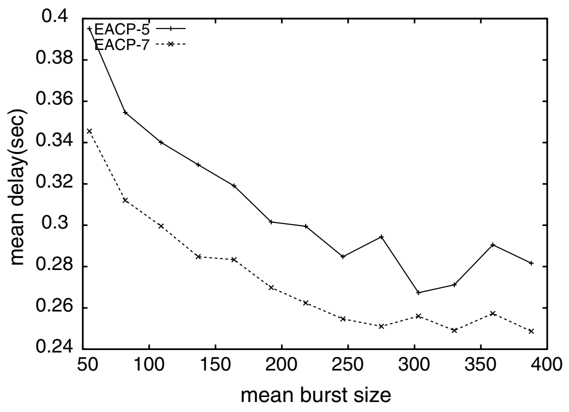

Fig. 11. Burst size vs. mean delay for *EaPb*.

increases at higher burst size. This is due to the increase in the throughput at lower burst size and almost a constant at higher burst size shown in Fig. 5. For exponential inter-arrival time and fixed burst size, delay is plotted in Fig. 10. From Fig. 10 it is observed that delay decreases with increase in the burst size; this is due to the increase in the throughput as shown in Fig. 6. Identical observation is made for the exponential inter-arrival and Pareto distributed burst size shown in Fig. 11.

It is observed from Figs. 9–11 that with increase in the number of wavelengths the delay experienced by requests decreases. This is in accordance with the WDM technology that the delay experienced decreases with increase in the number of wavelength. As expected the delays experienced by high-priority requests are lower than the low-priority requests as shown in Fig. 12 for the case *EaPb*. Identical observations were made for other two cases – *DaDb* and *EaDb*.

Finally, we include the plot for the delay experienced with varying number of nodes at a given wavelength for exponential inter-arrival and Pareto distributed burst-size in Fig. 13 for the case *EaPb*. It is observed that there is decrease in delay; this is due to the increase in the throughput with increase in the number of nodes as shown in Fig. 8. Thus all the results are consistent.

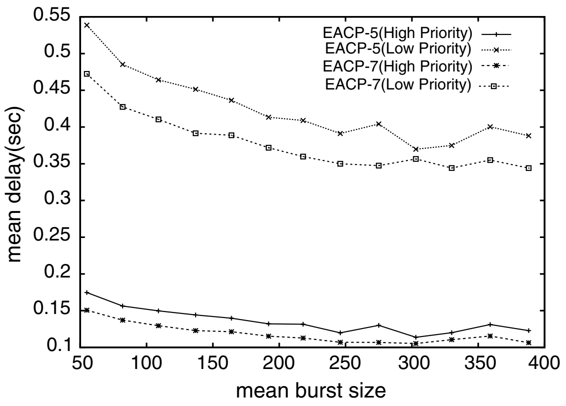

Fig. 12. Burst size vs. mean delay of high and low priority for *EaPb*.

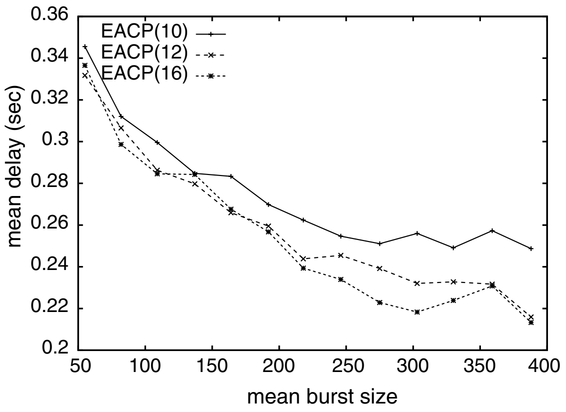

Fig. 13. Burst size vs. mean delay for *EaPb* for nodes 10, 12 and 16 and seven wavelengths.

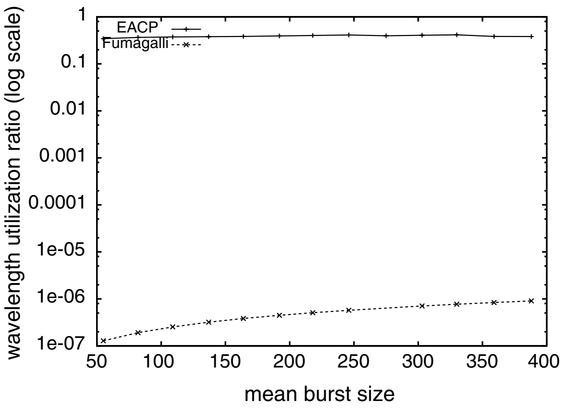

Fig. 14. Burst size vs. wavelength utilization for *EaPb*.

5.2. Comparison of EACP algorithm with Fumagalli et al. [4]

In Fig. 14, we plot the wavelength utilization by EACP algorithm and the algorithm proposed by Fumagalli et al. It is observed from Fig. 14 that the wavelength utilization in our proposed algorithm is much higher than that of Fumagalli et al.'s scheme. The lower wavelength utilization in their algorithm is mainly due to the way it releases the lightpaths. For every request served, the wavelength reserved by the node remains un-utilized for at least one or at the most two token cycles giving lower utilization in their scheme. This is not the case with EACP.

The delay experienced by packets in their scheme is lower than our algorithm as shown in Fig. 15. This is mainly due to the difference in node architecture. Their nodes are equipped with an array of fixed transmitters and receivers. Thus, a node can transmit and receive more than one data-channels at the same time. In our algorithms, nodes have a single tunable transceiver each where a single transmission or/and reception *alone* can take place at the same time. We believe that with the recent advances in laser technology, nodes will be equipped with tunable transceiver rather than a fixed array of transmitters and receivers in future.

In Fig. 16, we compare the throughput, in *bps*, for their scheme with our algorithm. We observed that the throughput (*bps*) obtained by their scheme is higher than our algorithm; this is because of the array of

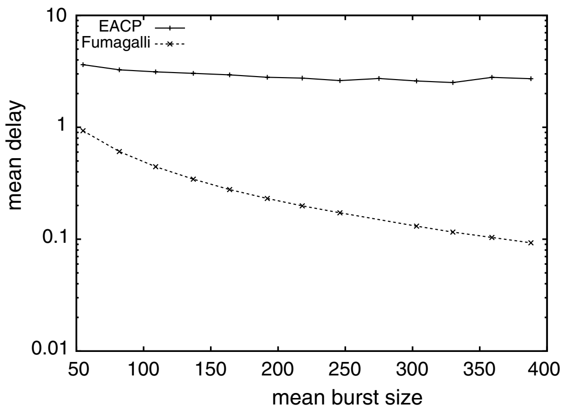

Fig. 15. Burst size vs. mean delay for *EaPb*.

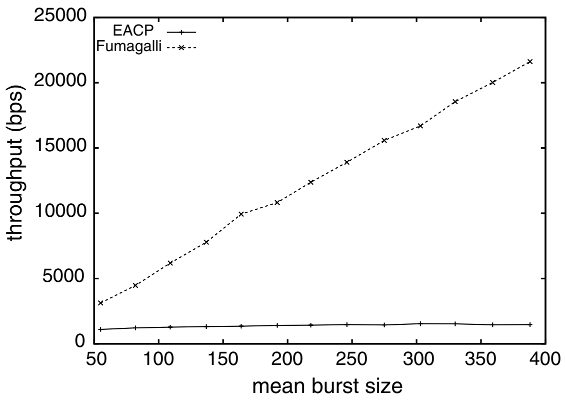

Fig. 16. Burst size vs. throughput (bps) for *EaPb*.

transmitters and receivers used by them whereas we used a single tunable transceiver. In Fig. 15, it is observed that the mean delay in their case decreases with increase in burst size and there is a corresponding increase in throughput (bps) with increase in burst size as shown in Fig. 16.

The main disadvantage of Fumagalli et al.'s scheme is the poor scalability – their scheme does not scale with wavelength. With increase in the number of wavelengths, the number of transmitters and receivers have to be increased proportionally. Second, with increasing number of wavelengths, the cost increases due to addition of transmitter(s) and receiver(s). Third, it also incurs higher maintenance cost because of multiple tokens in the ring. Our scheme is scalable, it does not incur any additional equipment cost or the maintenance cost with increase in wavelengths; we do not need any additional tokens too. Further, it is difficult to incorporate the notion of QoS in their scheme.

6. Conclusions

In this paper, we have proposed a node architecture and an EACP algorithm to access the shared medium and to provide QoS in an optical ring network. As desired in any MAC protocol the EACP algorithm is contention-free. We have observed that the throughput increases with increase in number of wavelengths and also with increase in number of nodes. As expected, the higher priority requests experience lesser delays than the low priority requests. The proposed node architecture uses a tunable transceiver for data-channel and a fixed transceiver for control channel. The tunable transceiver is an emerging device and is in tune with the advances in laser technology, and makes the scheme scalable. This proposal is an extension of our recent work [1] and includes the QoS provisioning.

We compared our scheme with that of Fumagalli et al. [4]. Though their scheme does not support QoS, nonetheless, this is closest to our work. We found that our scheme performs better in terms of wavelength utilization. Their scheme has smaller delays and higher throughput (bps); this is mainly due to the difference in node architecture. The node architecture in our scheme has only one tunable and fixed transceiver each, while that of their scheme has an array of transmitters and receivers. The use of tunable components in our scheme is in tune with the recent advances in laser technology. It is widely believed that, in future, nodes in WDM networks will be equipped with tunable transceiver over an array of fixed transmitters and receivers, however, at a little additional cost.

References

[1] A.K. Turuk, R. Kumar, R. Badrinath, Opt. Commun. 231 (1–6) (2004) 199.
[2] R. Ramaswami, K.N. Sivarajan, Optical Networks: A Practical Perspective, Morgan Kaufmann, Los Altos, CA, 1998.
[3] C.S.R. Murthy, G. Mohan, WDM Optical Networks: Concepts, Design and Algorithms, Prentice-Hall of India Limited, 2000.
[4] A. Fumagalli, J. Cai, I. Chlamtac, in: IEEE Globecom, Sidney, November, 1998.
[5] A. Fumagalli, J. Cai, I. Chlamtac, in: Proceedings of the ICC'99 Conference, Vancouver, Canada, June, 1999.
[6] J. Fransson, M. Johansson, M. Roughan, L. Andrew, M.A. Summerfield. Available from: <http://citeseer.nj.nec.com/487875.html>.
[7] M.A. Marsan, A. Bianco, E. Leonardi, A. Morabito, F. Neri, IEEE Commun. Mag. (1999) 58.
[8] K. Bengi, H.R. van As, IEEE JSAC 20 (1) (2002) 216.
[9] M.A. Marsan, A. Bianco, E.G. Abos. Available from: <http://citeseer.nj.nec.com/34986.html>, 1996.
[10] M.A. Marsan, A. Bianco, E. Leonardi, F.Neri, S. Toniolo. Available from: <http://citeseer.nj.nec.com/marsan97almost.html>, 1997.
[11] D.A. Levine, I.F. Akyildiz, IEEE/ACM Trans. Network. 3 (2) (1995) 158.
[12] F. Jia, B. Mukherjee, J. Lightwave Technol. 11 (5/6) (1993) 1053.
[13] B. Mukherjee, IEEE Network (1992) 12.
[14] P.E. Green, L.A. Coldren, K.M. Johnson, J.G. Lewis, C.M. Miller, J.F. Morrison, R. Olshansky, R. Ramaswami, E.H. Smith, J. Lightwave Technol. 11 (5/6) (1993) 754.

[15] L.G. Kazovsky, P.T. Poggiolini, J. Lightwave Technol. 11 (5/6) (1993) 1009.
[16] O.K. Tonguz, K.A. Falcone, J. Lightwave Technol. 11 (1993) 1040.
[17] I. Chlamtac, A. Ganz, in: Proceedings of the IEEE ICC'89, vol. 2, 1989, p. 739.
[18] I.M.I. Habbab, M. Kavehrad, C.E.W. Sundberg, J. Lightwave Technol. 5 (1987) 1782.
[19] N. Mehravari, J. Lightwave Technol. 8 (1990) 520.
[20] G.N.M. Sudhakar, N.D. Georganas, M. Kavehrad, J. Lightwave Technol. 9 (10) (1991) 1411.
[21] http://zdnet.com.com/2102-1103-986221.html.
[22] F. Jia, B. Mukherjee, J. Iness, IEEE/ACM Trans. Network. 3 (1995) 477.
[23] C.-K. Chan, L. kuan Chen, K.-W. Cheung, IEEE JSAC 14 (5) (1996) 1052.
[24] O.A. Lavrora, G. Rossi, D.J. Blumenthal, in: Proceedings of the ECOC 2000, vol. 2, September 2000, p. 169.
[25] B. Mason, G.A. Fish, S.P. Denbaars, L.A. Coldren, IEEE Photon Technol. Lett. 11 (1999) 638.
[26] C.R. Doerr, C.H. Joyner, L.W. Stulz, IEEE Photon Technol. Lett. 11 (1999) 1348.
[27] W. Cho, B. Mukherjee, in: Global Telecommunications Conference, 2001, GLOBECOM'01, IEEE, vol. 3, 25–29 November, 2001, p. 1575.
[28] A. Leon-Gracia, I. Widjaja, Communication Networks: Fundamental Concepts and Key Architecture, Tata McGraw-Hill Publishing Company Limited, 2000.
[29] O.K. Tonguz, K.A. Falcone, J. Lightwave Technol. 11 (5/6) (1993) 1040.
[30] V. Paxson, S. Floyd, IEEE/ACM Trans. Network. 3 (3) (1995) 226.
[31] T.D. Neame, M. Zukerman, R.G. Addie, in : Proceedings of Broadband Communication'99, Hong Kong, 10–12 November, 1999, p. 73.