QoS Provisioning in WDM Ring Networks with Tunable Transceivers

Ashok K. Turuk and Rajeev Kumar*
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur
Kharagpur, WB 721 302, INDIA
{akturuk, rkumar}@cse.iitkgp.ernet.in

Abstract – In this paper, we propose two algorithms and a node architecture to access the shared medium and to support priority based quality-of-service (QoS) in WDM ring network. The algorithms give priority to a node having high priority request in reserving a destination node and/or a data-channel. The first algorithm selects a data-channel based on earliest availability of the data-channel, and the second with minimum scheduling latency. Both the schemes employ reservation mechanism, however they differ from the traditional reservation mechanism in that no explicit release of the reserved resources take place. The proposed schemes are contention-free, easy to implement and require no buffers at nodes. The proposed node architecture is configured around tunable transceiver(s) and thus makes the scheme scalable. We study performance of both the schemes, by simulation, for bursty traffic modeled using an M/Pareto distribution, and compare the performance with another token based algorithm. We found that our schemes perform better in terms of wavelength utilization. However, delays are higher due to the single tunable transceiver used in our scheme as opposed to an array of transmitters and receivers in the previous work.

Keywords – quality of service, optical ring network, priority, WDM, medium access control, tunable transceiver.

*corresponding author

1 Introduction

It is widely believed that the increasing demand for bandwidth at the backbone network due to the rapid increase in the number of Internet users as well as the growing Internet applications of voluminous data can be satisfied by the WDM technology. The end-users of Internet applications are mostly hooked to a Local Area Network (LAN). With increase in demand for bandwidth at LANs the demand for bandwidth at backbone network has increased proportionately. To meet the bandwidth requirements at LANs, much work is being reported in the literature for the deployment of WDM technology in LANs. A growing number of Internet applications require some kind of prioritized quality-of-service (QoS) guarantee such as delay constraints and low packet loss. To provide end-to-end QoS, LANs must support some kind of QoS. Bandwidth available in a LAN is shared among all the network users, therefore, 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 LANs based on star or ring as the underlying physical topology [1, 2].

In this paper, we focus on a LAN based on ring topology. Ring offers several attractive features such as higher channel utilization and bounded delay. There are three well-known access strategies for LANs based on ring topology, namely, *token ring*, *slotted ring*, and *insertion ring*. These have been widely used as LANs both in commercial systems and research prototypes. WDM slotted rings are reported in [3 - 6] where synchronization among the slots are done at WDM ADMs [7]. Token based WDM ring network is explained in Fumagalli *et al.* [8, 9]. Unlike *FDDI* rings, Fumagalli *et al.* [8, 9] discussed multiple tokens in a ring.

Most of the MAC protocols for WDM ring network are based on the case where there are as many wavelength channels as the nodes in the network. For example, in Fumagalli *et al.*'s [8, 9] mechanism, the number of tokens, number of transmitters and receivers at each node in the ring are equal to the number of data-channels. This results in scalability problem. Most of the proposed protocols equip their nodes with either tunable transmitter and fixed receiver (TT-FR) or fixed transmitter and fixed receiver (FT-FR). A few of them [3, 4, 9, 10] require an array of transmitters and/or receivers at each node and, thus add to high equipment cost. Moreover, such architectures are not scalable.

Additionally, when operating in a multi-traffic environment a MAC protocol should be able to interleave the different traffic types 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 WDM ring network; for the representative work done in this area, see [3, 4, 8]. Marsan *et al.* [4] proposed an incremental slot reservation strategy based on the local node traffic for the *TT-FR* architecture. Bengi *et al.* [3] proposed different access strategies for real-time and non-real-time data traffic. Fumagalli *et al.* [8] proposed a control channel based multi-token approach to provide QoS. While proposals [3, 4] are for slotted ring, proposal [8] is for token based ring. the number of channels is equal to the number of nodes in [4], while in [3] the number of nodes is greater than the number of channels. In [8], each nodes is equipped with a transceiver-array imposing constraints on scalability.

In this paper, we propose a node architecture and two variants of token based algo-

rithms to access the shared medium and to support QoS in optical ring networks. In our proposal, each node is equipped with one tunable and one fixed transceiver each. Both the algorithms presented in this paper differ in selecting a data-channel. First algorithm which we call *earliest available channel with priority (EACP)* selects the earliest available data-channel. (The EACP algorithm is a variant of our previous EAC algorithm [11] which does not include priority.) Second algorithm which we call *minimum scheduling latency with priority (MSLP)* selects a data-channel with minimum scheduling latency. We define minimum scheduling latency as the difference in time between start of the transmission on the channel and availability of the channel. We have shown in this work, by simulation, that the minimum scheduling latency increases the channel throughput and decreases the delay. Both the proposed algorithms operate in a distributed manner and are based on reservation. However, both differ from the traditional reservation schemes in that no explicit release of the reserved resources takes place. To overcome high reservation latency, transmission of a node is overlapped with reservation. We have shown that both the algorithms have the capability of avoiding channel collisions and destination conflicts. We used bursty traffic for simulating the performance. We compare the performance of *EACP* algorithm with another token based algorithm proposed by Fumagalli *et al.* [8] which also supports QoS in a WDM ring network and is closest to our work.

The rest of the paper is organized as follows. Architecture and notations used in the system model are described in Section 2. In Section 3, the proposed algorithms are detailed. Correctness of algorithms is illustrated in Section 4. Simulation results are presented in Section 5. Finally, conclusions are drawn in Section 6.

2 System Model

There exist few node architectures for WDM ring networks. For example, Cho and Mukherjee [12], proposed a node architecture for WDM ring networks where each node is equipped with a SONET ADM and a DWADM (Dynamic Wavelength Add-Drop Multiplexers). In their architecture a node must transmit and receive on the same wavelength. This limits usage of the wavelength and hampers performance of the ring. Fumagalli *et al.* [8, 9] proposed a node with an array of fixed transmitters and receivers. Fixed transmitters and receivers limit the scalability of the network. Some work is reported for nodes with tunable transmitters [13 - 15], and with tunable receivers [16 - 22]. To overcome limitations of [12] and scalability of [8, 9], we propose an architecture for a node in WDM ring networks as shown in Fig. 1, and discuss the node-architecture in subsequent sub-sections.

2.1 Architecture

We assume N number of nodes in the network, numbered as 0,1,2, ..., $N-1$. Node i of the network is connected to node j by an optical fiber such that $j = (i+1) \bmod N$ where $i \neq j$, and for all i, $j \in 0,1, \ldots, N-1$ then node j is the successor of node i and node i is the predecessor of node j. The system supports W wavelengths λ_0, λ_1, ... , λ_{W-1} out of which $(W-1)$ number of channels are data-channels and one control channel. One of the wavelengths, say, λ_0 is dedicated to control channel and rest of the wavelengths

are used for data-channels. A circuit is established on wavelength, λ_0, between every pair of adjacent nodes i and j. The circuit thus established uses the dedicated control channel. A pair of nodes i and j are said to be adjacent if j is successor of node i and node i is predecessor of node j.

In this work each node is equipped with a single *fixed* transceiver for control channel and a *tunable* one for data-channels. The fixed transmitter and receiver are tuned to wavelength, λ_0, to transmit and receive control information between the adjacent nodes, and tunable transmitter and receiver are tuned to data-channels as and when required. Each node in the network maintains status of its transmitter, receivers of other nodes, and data-channels in the network. Status gives the time at which transmitter, receivers, and data-channels are available. For two nodes to communicate, tunable transmitter of the source and tunable receiver of the destination must be tuned to the same wavelength (data-channel). (Information transfer now takes a single hop over a data-channel.) With the advances made in laser technology, It is predicted by many that, in future, nodes of WDM networks will be equipped with tunable lasers. With a little additional cost, tunable lasers offer several attractive features such as they can be remotely programmed to adjust to the changing conditions, and no replacement of lasers is needed if spectrum requirements change [23].

There is a *single* token which circulates around the ring on the control channel. The token consists of N fields, each field we call a *slot*; *slot* l is assigned to node l. Each field is subdivided into *seven* mini-fields which we collectively call *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 the token at each node and R is the Ring Latency. Since TP is same for all the nodes in the network each node gets a fair chance to access the shared medium. Thus, the delay involved is bounded which is the inherent feature of a ring network.

A node on receiving the token processes each slot, l, $(0 \leq l < N)$ 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 a data-channel. A node reserves the destination and a data-channel by writing the control information at its allotted slot in the token. Reservation mechanism is explained in Section 3. A node i writes control information in slot i of the token and modify slot j if reservation request of node j is de-reserved at node i. A node has $N-1$ buffers, one for each destination node. Additionally, each node has *send*, *receive*, *interrupt*, and *token processing* modules; *req_rec*, *req_send* and req_rec' queues; DAT, CAT, DAT' and CAT' vectors. The functionality of each of these elements is explained in the following section.

2.2 Notations and Definitions

Each node i maintains four vectors namely, 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

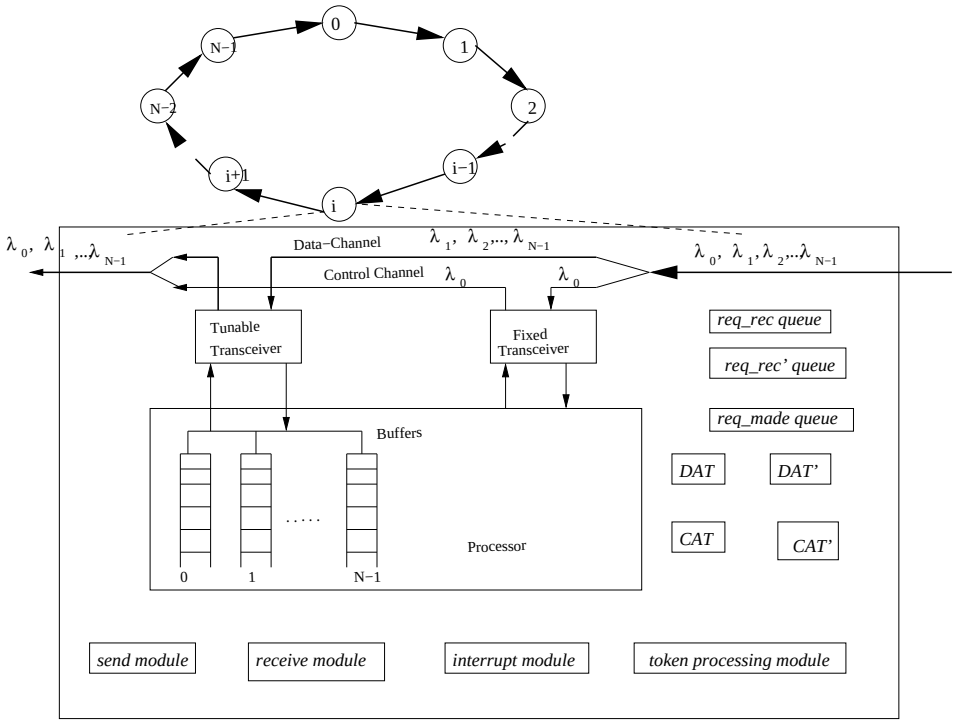

Figure 1: Architecture of a Node i in a Ring Network

DAT' and CAT' are copies of the 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' : Sets of nodes whose requests are de-reserved at node i, and
$current\_time$: time at which an action is taken at the node.

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, and 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 which is 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 its *req_rec* queue. Elements of *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 node j is de-reserved or node j is not making request for reservation. When s is *zero* the $slot_j$ will not be processed by the node.

d : identity of the destination node requested for reservation.

c : identity of the data-channel requested for reservation.

t_c : time at which transmitter of the source and 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, and tt - tuning time of the transmitter and the receiver in the modified request. Initially all sub-fields are set to negative values.

3 Algorithms

In this section, we detail our proposed EACP and MSLP algorithms. Both the algorithms are based on reservation mechanism. However, they differ from the traditional reservation mechanism in the following aspects. First, no explicit release of the reserved resources takes place. Second, unlike the traditional reservation mechanism where resources are reserved only when they are free, our proposed algorithms looks ahead to find the time at which the required resources are available and reserves the resources from that point of time. 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. The reserved resources can be requested for reservation by another node after the requested period. This does not necessitate an explicit release of the reserved

resources. Two nodes can make reservation requests for the same resources during the same token cycle but for different times. 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 and remains in place until the transmission is completed. Availability of fast tuning lasers reported in [24 - 27] makes it possible to set up lightpaths dynamically; this is in line with the upcoming technology.

Both the algorithms give priority to a node having a 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 a high priority request de-reserves it. A low priority request is de-reserved at a node by setting s field of the *slot* corresponding to that request to *zero*. De-reserved requests are not processed at subsequent nodes. To avoid starvation of low priority requests, the priority of such requests is upgraded to a high after the failure of certain number of reservation attempts. A request whose priority has been changed to high is treated as a new request to the node, so that the previously arrived high priority requests are served first. Each period of transmission is determined at the time reservation request is made. Duration of transmission for which a request is made is monitored at each node for the last TP.

Both the algorithms have the following modules: *Send*, *Receive*, *Token processing*, and *Interrupt* modules. While *Send*, *Receive*, and *Interrupt* modules are identical for both the algorithms, the algorithms differ in *Token processing* module. The function of each of the modules is explained below.

Send module: If a node's *req_made* queue is non-empty and the transmitter, T_x, of the node is $FREE$, then the node invokes its *send* module. The front element of the *req_made* queue is removed and the status of the transmitter, T_x, is set to BUSY. The transmitter is tuned to the data-channel specified in the *dc* field of the removed element at the time specified in the *tt* field of that element. Once the transmitter of the node is tuned to the specified data-channel, transmission from the node begins and continues for a duration specified in the *td* field of the element. After transmission is completed, status of the transmitter, T_x, is set to $FREE$, and another transmission process begins. Note the transmitter of the node is tuned to the specified data-channel only at the specified time even-though status of the transmitter is set to $BUSY$. Successful reservation requests are added in the *req_made* queue in the order in which the successful requests are made.

Receive module: Analogous to the *send* module, if a node's *req_rec* queue is non-empty and receiver, R_x, of the node is $FREE$, then the node invokes its *receive* module. The front element of the *req_rec* queue is removed and status of the receiver, R_x, is set to $BUSY$. Receiver is tuned to the data-channel specified in the *dc* field of the removed element at the time specified in the *tt* field of that element. Once the receiver of the node is tuned to the specified data-channel, node starts receiving information and continues for a duration specified in the *td* field of the element. After the transmission

is completed, status of the receiver, R_x, is set to $FREE$, and another transmission process begins. Note receiver of the node is tuned to the specified data-channel only at the specified time even-though status of the receiver is set to $BUSY$.

Interrupt module: A reservation request destined to a node is added in *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. Following actions are taken by the *token processing* module. First, if the request made by the node is successful then following actions take place: it sets the s field of its own slot to *zero*; it updates the available time of its transmitter, destination node receiver and the reserved data-channel; it adds the request to its *req_made* queue; it sets the rm vector of its own slot to a negative value. Second, all high priority requests are checked to find if any high priority request has modified any low priority request. If such a request exists, the values of DAT and CAT vectors modified by the low priority request are set to the previous values. If the modified low priority request exists in the *req_rec* queue of the node then it is deleted from the queue. If the *receive* module has removed the request from the *req_rec* queue then *Interrupt* module is invoked. Third, values of DAT and CAT vectors are updated. If the buffers are non-empty, then a high priority burst is selected if it exists else a low priority burst is selected. Then it finds the destination identity of the burst which has the maximal waiting time, and the earliest available data-channel in case of EACP and a data-channel with minimum scheduling latency in case of MSLP 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 its 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 its reservation request is completed i.e., on or after $t + TP$. Therefore, if $t^{'} < t + TP$ the value of $t^{'}$ is set to $t^{'} = t + TP$. 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 subsections, we write pseudocode for both the algorithms.

3.1 *EACP - Algorithm*

Perform the following *Cases* at each node i

CASE: **if** (*req_made* queue is non-empty and $T_x = FREE$) invoke *Send* module.

CASE: **if** (*req_rec* queue is non-empty and $R_x = FREE$) invoke *Receive* module.

CASE: Invoke *Token processing* module when *node* i receives the token.

3.1.1 *Send module*

1. Remove the front element of the *req_made* queue. Let it be $req\_made(l)$.
2. $T_x \leftarrow BUSY$
3. **if** $(current\_time \geq req\_made(l) \cdot tt)$
   - Tune the transmitter to data-channel, $req\_made(l) \cdot dc$
4. **if** $(current\_time \geq req\_made(l) \cdot tt + t_u)$
   - Transmit data to node, $req\_made(l) \cdot di$
5. **if** $(current\_time \geq req\_made(l) \cdot tt + t_u + t_p + req\_made(l) \cdot td)$
   - $T_x \leftarrow FREE$

3.1.2 *Receive module*

1. Remove the front element from the *req_rec* queue. Let it be $req\_rec(l)$. Add a copy of it to $req\_rec^{'}$ queue.
2. $R_x \leftarrow BUSY$
3. **if** $(current\_time \geq req\_rec(l) \cdot tt)$
   - Tune the receiver to data-channel, $req\_rec(l) \cdot dc$
4. **if** $(current\_time \geq req\_rec(l) \cdot tt + t_u)$
   - Receive data from the source
5. **if** $(current\_time \geq req\_rec(l) \cdot tt + t_u + t_p + req\_rec(l) \cdot td)$
   - $R_x \leftarrow FREE$

3.1.3 *Interrupt module*

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

3.1.4 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.
   - $DAT[d] \leftarrow CAT[c] \leftarrow DAT[i] \leftarrow t_c + t_u + t_p + D$

     [The values of DAT and CAT vectors are updated for successful reservation requests]

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 $(rm[k].sn \geq 0)\{$

     – switch$(rm[k].sn \geq 0)\{$

     ∗ case 0 or $N - 1$:
     if $(i \geq mod(rm[k].sn + 1, N)$ and $(i \leq mod(j - 1, N))\{$

     · $DAT[rm[k].dn] \leftarrow DAT^{'}[rm[k].dn]$

     · $CAT[rm[k].wc] \leftarrow CAT^{'}[rm[k].wc]$

     · **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 $(i \geq mod(rm[k].sn + 1, N)$ or $(i \leq mod(j - 1, N))\{$

     · $DAT[rm[k].dn] \leftarrow DAT^{'}[rm[k].dn]$

     · $CAT[rm[k].wc] \leftarrow CAT^{'}[rm[k].wc]$

     · **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

     [Low priority requests that are de-reserved by high priority requests are either removed from *req_rec* queue of the node if exist or the transmission corresponding to the requests is dropped.]

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** ($p = 1$)
     - $DAT^{'}[d] \leftarrow DAT[d]$
     - $CAT^{'}[c] \leftarrow CAT[c]$

     [Values of DAT and CAT vectors are updated for all requests and the values of $DAT^{'}$ and $CAT^{'}$ vectors are updated for high priority requests.]

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 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_λ
   - 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 .
   - $DAT[x] \leftarrow DAT^{'}[x]$, $CAT[c] \leftarrow CAT^{'}[c]$

     [Low priority request for destination node x is de-reserved. Values of DAT and CAT vectors are restored to their previous values.]

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 $m \leftarrow 1, \cdots, W - 1$ $\}$.

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 .
    - $DAT[x] \leftarrow DAT^{'}[x]$, $CAT[c] \leftarrow CAT^{'}[c]$

[Low priority request for data-channel k is de-reserved. Values of DAT and CAT vectors are restored to their previous values.]

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 .
      - $DAT[x] \leftarrow DAT^{'}[x]$, $CAT[c] \leftarrow CAT^{'}[c]$

    [De-reserve all requests that are related to a low priority request that is de-reserved for want of destination node.]

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 .
      - $DAT[x] \leftarrow DAT^{'}[x]$, $CAT[c] \leftarrow CAT^{'}[c]$

    [De-reserve all requests that are related to low priority requests; such requests are de-reserved for want of data channel.]

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 data channel k are available at the same time.

17. **if** $(\tau < current\_time + TP)$ $\tau = current\_time + TP$

18. Calculate 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 x.

21. Find the earliest available data-channel $k \leftarrow \{m : CAT[m]$ is minimum for $m \leftarrow 1, 2, ..., W-1$ $\}$.

22. $\tau_t \leftarrow DAT[i]$, $\tau_d \leftarrow DAT[x]$, $\tau_c \leftarrow CAT[k]$

23. $\tau \leftarrow max(\tau_t, \tau_d, \tau_c)$

24. **if** $(\tau < current\_time + TP)$ $\tau = current\_time + TP$

25. Calculate 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.

28. Send the token to the successor node.

3.1.5 Simulation of EACP Algorithm

We illustrate the reservation process in EACP algorithm with the following example. For simplification, we consider a 4-node ring network and two data-channels. Available time of the transmitter and the receiver of each node, and data-channels is shown in Table-1. Traffic at each node, at some point of time t, is shown in Fig. 2, and the same is tabulated in Table-2. The entry $b_i(x, y)$ corresponding to row m and column n of Table-2 indicates that source m has a burst of $i - th$ 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 $TP = 20$ and $t_p = 10$ (computed as in [12]).

Node	0	1	2	3
Transmitter	45	40	110	47
Receiver	40	47	45	110
CAT[λ_1] = 47,		CAT[λ_2]= 110		

Table 1: Available time of transmitters and receivers of nodes, and data-channels.

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 2: Traffic (as depicted in Figure 2) at different nodes in tabular form.

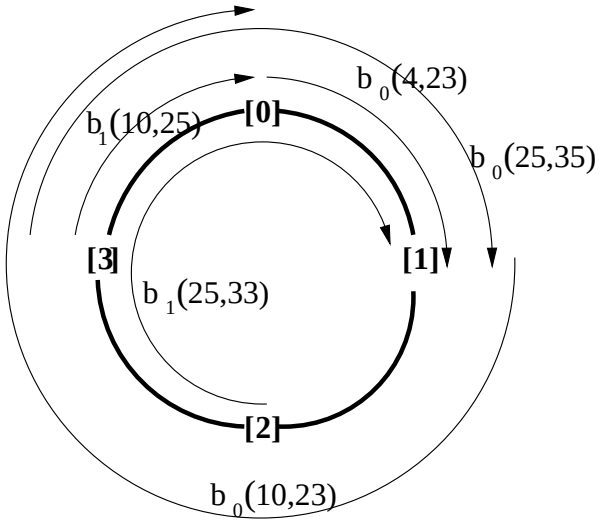

Figure 2: Illustration of traffic (as given in Table 2) at different nodes in the ring.

Let $t = 40$, and node 0 receives the token at time 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 (τ_t, τ_d, τ_c) i.e., 47. $\tau < t + t_u + TP$ so 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 sent to node 1.

Node 1 updates values of DAT and CAT vectors at its node as 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.

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

Table 3: 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

Table 4: Contents of DAT and CAT vectors at node 1.

Node 1 selects 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 sent to node 2.

Node 2 selects 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 request data-channel λ_1 which is selected by node 2 having a high priority request). 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. Values of DAT and CAT vectors at node 3 after processing the token are shown in Table-6.

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

Table 5: Contents of DAT and CAT vectors at node 2.

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

Table 6: Contents of DAT and CAT at node 3.

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 not successful, and makes another reservation attempt. Values of DAT and CAT vectors after processing the token are shown in Table 7.

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

Table 7: Contents of DAT and CAT vectors at node 0.

Request from node 3 is entered in req_rec of node 0. Node 0 selects destination node 2 and data-channel λ_2, and performs the request operation as explained earlier. When node 1 receives the token values of $DAT[2]$ and $CAT[\lambda_1]$ elements 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. 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 req_rec queue of node 1.

Reservation process continues as explained above. Before a node updates values of its DAT and CAT vectors copies of the vectors are stored in $DAT^{'}$ and $CAT^{'}$ vectors, respectively so that these can be restored back to the previous values if the

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

Table 8: Contents of DAT and CAT vectors at node 1.

request becomes un-successful. In the above example, we have illustrated working of the algorithm for node 1 only.

3.2 MSLP Algorithm

In EACP algorithm, a node selects the earliest available data-channel. This may give rise to higher scheduling latency and lower wavelength utilization, if the destination node is available at a time much later than the data-channel availability. To reduce the scheduling latency and give higher wavelength utilization we propose the MSLP algorithm that selects a data-channel with minimum scheduling latency. We expect this strategy to give rise to higher wavelength utilization and lower delay in most of the cases. We illustrate, in the following paragraphs with an example, the difference between EACP and MSLP algorithms.

Suppose a node wishes to transmit and the destination is available at time d_t. Let the system has three channels $c1$, $c2$ and $c3$ and they are available at time t_{c1}, t_{c2} and t_{c3}, respectively such that the following condition $t_{c1} < t_{c2} < t_{c3} < d_t$ holds. In EACP algorithm, the node selects the earliest available data channel $c1$ and the scheduling latency on the data channel is $d_t - t_{c1}$. In MSLP algorithm, the minimum scheduling latency channel $c3$ is selected for transmission. The scheduling latency on channel $c3$ is $d_t - t_{c3}$ which is less than the scheduling latency on channel $c1$.

Next, we illustrate another situation where MSLP has advantage over EACP. Let us consider a situation where node a wishes to communicate with node c, and node b wishes to communicate with node d. Let t_c and t_d be the time at which destination nodes c and d, respectively will be available. Suppose the following condition $t_{c1} < t_d < t_{c2} < t_{c3} < t_c$ holds and node a receives the token. In EACP, node a selects the earliest available data channel $c1$. When node b receives the token it selects data channel $c2$. In this case, the transmission from node b has to be delayed till data channel $c2$ is available. However, in MSLP the node a selects minimum scheduling channel $c3$, and node b selects channel $c1$, thereby reducing the transmission delay from node b; thus giving rise to better channel utilization.

Working of both ECAP and MSLP algorithm remains the same other than the channel selection strategy. MSLP algorithm selects minimum scheduling latency data-channel in step 10 and 21 of the token processing module. Except steps 10 and 21 of the *Token processing* module, MSLP algorithm remains exactly the same as that of EACP algorithm.

4 Correctness of the Algorithm

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

For the correctness illustrations for the cases (i) – (v), see [11]. The remaining illustrations for cases (vi) and (vii) are given below.

4.1 Low priority requests that are unsuccessful in making reservation and are added in the *req_rec* queue of the destination node are either removed from the *req_rec* queue or are 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. In the rest of this sub-section, we show that the request of node x is either not processed, or removed from the *req_rec* queue of node y. We consider the following two cases:

Case 1: The *receive* module of node y has removed the request of node x from *req_rec* queue of 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. 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 *req_rec* queue of 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-reserved 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 that node 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 previous values, 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. Let λ_1 be the selected data-channel. The intermediate nodes between nodes s_1 and s_2 (both exclusive) update values of $DAT[d]$ and $CAT[\lambda_1]$ elements at their nodes when they receive the token as mentioned in step 4 of the *token processing* module. A node makes a copy of the values of DAT and CAT vectors at its node in $DAT^{'}$ and $CAT^{'}$ vectors, respectively before updating the values of DAT and CAT vectors. 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. Both the algorithms give priority to node having high priority request in reserving the destination node. Node s_2 records the de-reserved 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-reserved request is not processed by other nodes.

When the intermediate nodes between s_1 and s_2 receive back the token values of $DAT[d]$ and $CAT[\lambda_1]$ elements are restored back to their previous values. This is done is step 1 of *token processing* module. Thus, values of DAT and CAT vectors updated at nodes due to de-reserved low priority requests are restored back to their previous values.

Case 2: Suppose nodes s_1 and s_2 have requests destined to nodes 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. Let λ be the selected data-channel. The intermediate nodes between s_1 and s_2 (both exclusive) update values of $DAT[d_1]$ and $CAT[\lambda]$ elements 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 values of DAT and CAT vectors at its node in $DAT^{'}$ and $CAT^{'}$ vectors, 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 nodes having high priority request in reserving a data-channel. Node s_2 records the de-reserved 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 values of $DAT[d_1]$ and $CAT[\lambda]$ elements are restored back to their previous values. This is done is step 1 of *token processing* module.

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

Case 3: Suppose requests of nodes i and j are related. If request of node i is de-reserved by a node z having high priority request, then 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 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 data-channels selected for reservation be λ_1 and λ_2, respectively. Suppose requests of nodes s_1 and s_2 are also related. Suppose node s_3 has high priority request to node d_3 and the data-channel selected be λ_1. When node s_3 receives the token values of $DAT[d_1]$, $CAT[\lambda_1]$ and $CAT[\lambda_2]$ elements are updated appropriately at the intermediate nodes. Node s_3 has a higher priority request, and selected data-channel is λ_1 which is also selected by node s_1 having a 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 request from node s_1 is de-reserved at node s_3. This results in de-reservation of the request from node s_2 at node s_3. This is done in step 14 *token processing* module.

The steps 12,13 and 14 of the *token processing* module de-reserve all the *related* requests. All de-reserved requests are recorded in rm vector of the control information field of the node that de-reserves the requests. In the case that we considered above, de-reserved requests are recorded in 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.

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 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 nodes d_1 and d_3 having higher priority requests). This is done in step 3 of the *token processing* module. Requests from nodes s_1 and s_2 are *related*, this results in de-reservation of the requests from node s_2. Steps 14, 15, and 16 de-reserve all the *related* requests. All de-reserved requests are recorded in rm vector of the control information field of the node that de-reserves the requests.

In the case that we considered above, de-reserved requests are recorded in 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 both the algorithms.

5 Simulation and Results

In this section we evaluate performance of the proposed node architecture and both the algorithms by simulation. We measure performance in terms of wavelength utilization, throughput, wavelength utilization and mean delay; these metrics are first class design parameters in WDM networks and have been used by many researchers. Wavelength utilization is the ratio of transmission time to transmission time plus scheduling latency. Scheduling latency is the difference between the time when a channel becomes free/idle and the time when transmission begins on the channel. Delay is the time for a request

to access the data channel. Since packets are not processed at intermediate nodes, so the time is establishing a lightpath is the sole contributor to delay. Therefore, delay is the time-difference between a request's arrival and start of its transmission.

For simulation, we take a 12 node ring network, and consider equally spaced nodes around the ring for simplicity. We vary number of data-channels from 6 to 8 for collecting simulation data. We use the following quanta of values in carrying-out the simulation: capacity of data-channel is 1 *Gbps*, length of the ring is 96 kms ($FDDI$ ring can span up to 200 kms [28]), processing time of token at each node is 1 μs, tuning time of transmitter and receiver is 5 μs (laser with tuning time of 5 ns are reported in the literature see [14 - 17]). Computed value of TP comes out to be 480 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]. We consider bursty traffic as the traffic in LANs are reported to be bursty in nature [30]. We consider both high priority and low priority traffic in our simulation. High priority traffic is generated with a probability of 0.4. We use M/Pareto distribution for generating bursts of both low and high priorities [31]. We keep the size of packets in each burst to be fixed at 10 kb per packet and the maximum size of the burst is kept at 48 packets.

For simulation we have considered the following three cases:

- Case 1: Both size and inter-arrivals of bursts are deterministic. We call this situation as *DaDb*.
- Case 2: Inter-arrival of bursts to follow an exponential distribution and deterministic burst size. We call this situation as *EaDb*.
- Case 3: Inter-arrival of bursts to follow an exponential distribution and burst size follows M/Pareto distribution. We call this situation as *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, and

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 real-world scenario neither the burst size nor the inter-arrival of burst are known in advance. Therefore real-world scenario is better modeled by *EaPb*; most of the results presented in the rest of this section belong to this case.

The rest of the section is divided in two parts. In first part, we include performance assessment for both the proposed EACP and MSLP algorithms. In second part, we

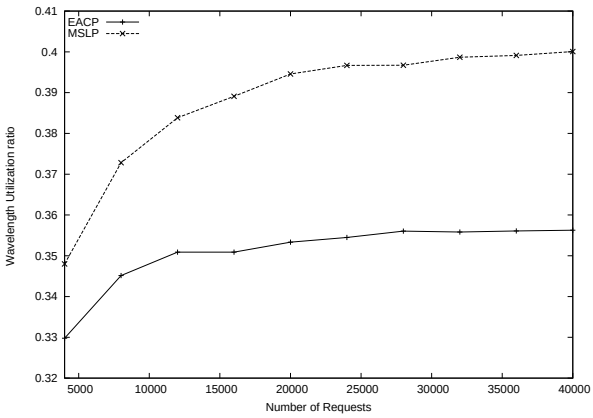

Figure 3: Wavelength utilization *vs.* number of requests for $EaPb$

include comparison of EACP algorithm with another algorithm which is closest to our work and was proposed by Fumagalli *et al.* [8].

5.1 Performance Estimation of EACP & MSLP Algorithms

A. Wavelength utilization *vs.* **number of requests:** First, we estimate wavelength utilization with increase in number of requests for the three cases $EaPb$, $EaDb$ and $DaDb$, respectively, and include plots in in Figures 3, 4 and 5, respectively. From the plots in Figure 3 for $EaPb$ case (exponential inter-arrival of the bursts and Pareto distributed burst size), we observe that wavelength utilization for both EACP and MSLP algorithms increases with increase in the number of requests.

With increase in number of requests the duration of transmission is increased and there are more requests waiting to be transmitted from a node. However, increase in wavelength utilization in EACP is marginal with increase in the number of requests. MSLP algorithm has higher wavelength utilization than EACP algorithm. The higher wavelength utilization in MSLP is due to the lower scheduling latency channel that it selects. Similar are the trends for exponential inter-arrival of bursts and deterministic burst size ($EaDb$) as shown in Figure 4.

However, for fixed inter-arrival of bursts and fixed burst size ($DaDb$), we have taken burst size to be of 30 packets and the inter-arrival of burst to be at 1 ms, for simulation. It is observed from Figure 5 that the wavelength utilization remains almost constant both for MSLP and EACP algorithms. The constant nature of the wavelength utilization is due to the proportionate increase in the scheduling latency with increase in the number of requests. Again, wavelength utilization is higher for MSLP algorithm than in EACP due to the reasons stated earlier.

B. Mean delay *vs.* **number of requests:** Next, we include plots for mean delay experienced by high priority requests for $EaPb$ in EACP and MSLP algorithms in Figure 6 for different number of wavelengths. We have assumed that a request is dropped if the resources are not available for transmission within 100 ms of its arrival. So the maximum delay experienced by a request is bounded by100 ms, and the minimum delay

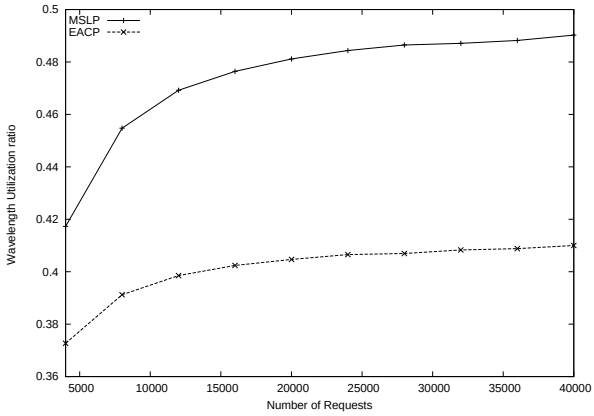

Figure 4: Wavelength utilization *vs.* number of requests for *EaDb*

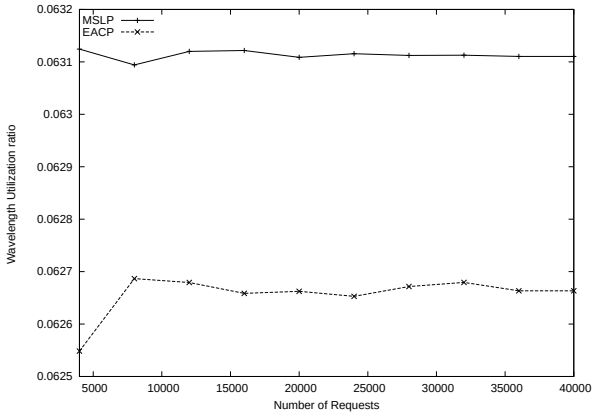

Figure 5: Wavelength utilization *vs.* number of requests for *DaDb*

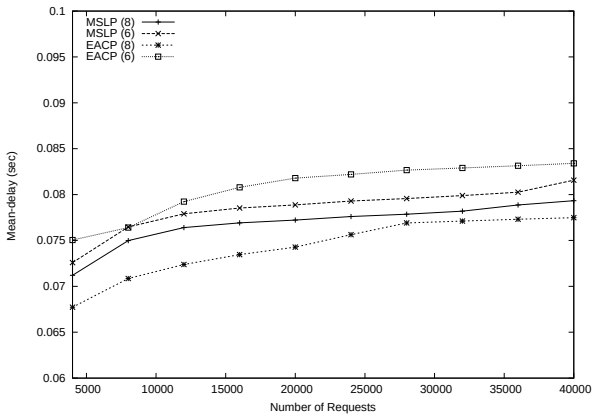

Figure 6: Mean delay *vs.* number of requests for high priority traffic in $EaPb$

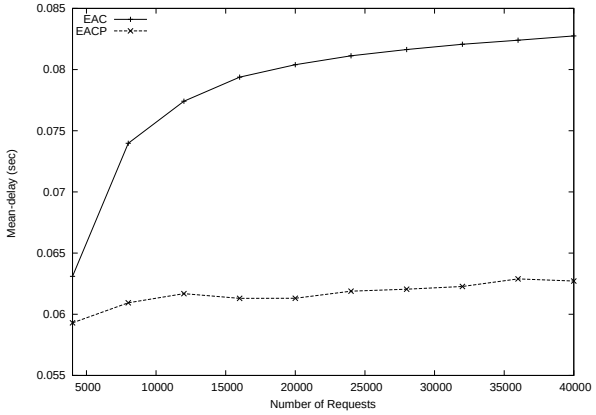

Figure 7: Mean delay *vs.* number of requests for EAC and EACP algorithms in $EaPb$

is that of a ring latency which is 480 μs in the present case. It can be observed from Figure 6 that requests experience lesser delays in MSLP than in EACP algorithm. The lesser delay in MSLP is due to the higher wavelength utilization as shown in Figure 3. It can also observed from the Figure that with increase in the number of data channel the delay decreases. This is in accordance with the WDM technology that the delay experienced decreases with increase in the number of data channels. Similar phenomenon is observed for the other two case $EaDb$ and $DaDb$.

To observe the efficacy of the priority based (EACPO and MSLP) algorithms over non-priority based (EAC) algorithm [11], we include plots, in Figure 7, of the mean delay experienced by high priority requests in EACP and the mean delay for (*any*) requests in EAC. The results clearly shows that the delay experienced by high priority requests in EACP is lower than the delay experienced by requests in EAC. This shows that the EACP gives priority to high priority request in transmission, and the priority-based algorithms work effectively.

C. Blocking probability *vs.* **number of requests:** Finally, we include plots, in Figure 8, for the blocking probability in EACP and MSLP algorithms for varying number

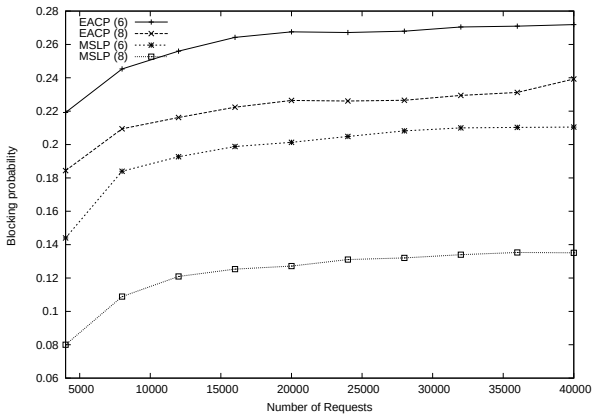

Figure 8: Blocking probability *vs.* number of requests for *EaPb*

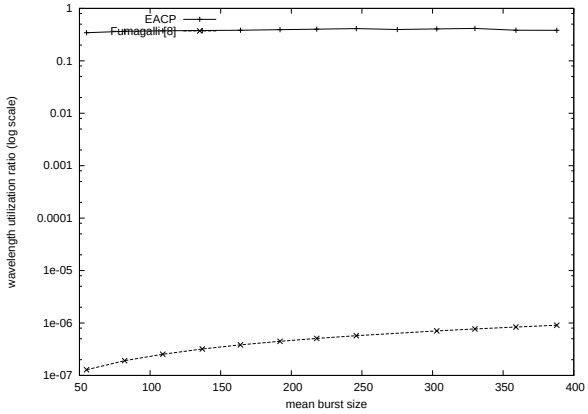

Figure 9: Wavelength utilization *vs.* burst size for *EaPb*

of data channels. We compute blocking probability as the fraction of the dropped requests over the overall requests. We have assumed that a request is dropped if resources are not available for transmission within 100 ms of its arrival. It can be observed from the figure that the blocking probability, in MSLP, is lower than that in EACP algorithm. The lower blocking probability in MSLP is attributed to the low delay experienced by overall requests in MSLP. We observe form the plots that overall blocking probability decreases with increase in the number of data channels. This phenomenon is in consistent with the WDM technology that the request-drop decreases with increase in the number of data channels.

5.2 Comparison of EACP algorithm with Fumagalli *et. al.* [8]

In this subsection, we give comparison between EACP and Fumagalli *et. al.*'s algorithm [8]. We consider bursts of varying sizes for comparison and assume that no burst is dropped. We include comparison results obtained for wavelength utilization, mean-delay and throughput. In Figure 9, we include plots for wavelength utilization in EACP algorithm and in the algorithm proposed by Fumagalli *et al.* [8]. It is observed from Figure 9 that the wavelength utilization in our proposed algorithm is much higher than

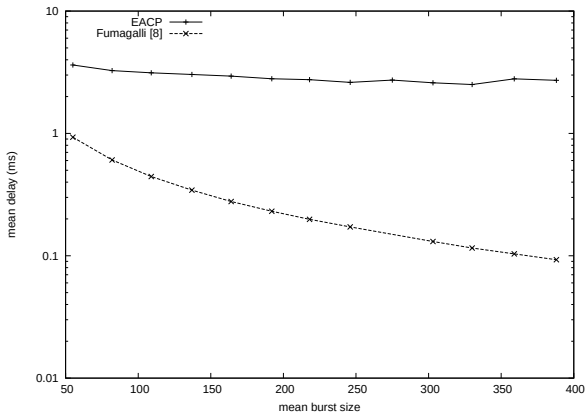

Figure 10: Mean delay *vs.* burst size for *EaPb*

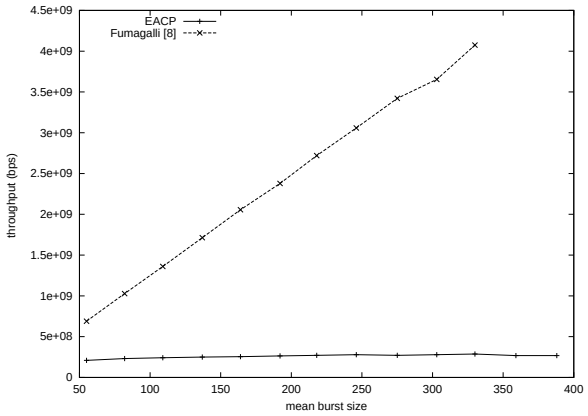

Figure 11: Throughput (bps) *vs.* burst size for *EaPb*

that of Fumagalli *et al.* The lower wavelength utilization in [8] is mainly due to the way it releases the lightpath. For every request served, the wavelength reserved by the node remains un-utilized for at least one or at most two token cycles and thus giving lower utilization in [8].

The delay experienced by packets in Fumagalli *et al.* is lower than our proposed algorithm as shown in Figure 10. This is mainly due to the differences in node architectures. In Fumagalli *et al.* [8], nodes are equipped with an array of fixed transmitters and receivers. Thus, a node can transmit and receive in more than one data-channels simultaneously. In our proposed algorithms, each node has a single tunable transceiver where only a single transmission and reception can take place at the same time, thus resulting in higher delays. It is widely believed that, with the advances in laser technology, the WDM nodes in future networks will be equipped with tunable transceiver(s) rather than a fixed array of transmitters and receivers.

In Figure 11, we plot throughput in *bps* for Fumagalli *et al.*'s algorithm [8] and our proposed EACP algorithm. We observe that the throughput(*bps*) for Fumagalli *et al.* [8] is higher than the one in our EACP algorithm for the reason stated above. In Figure 10, it is observed that the mean delay in Fumagalli *et al.* decreases with increase in burst

size and there is a corresponding increase in throughput; *bps* with increase in burst size is shown in Figure 11. This observation also confirms the fairness of the implementations of both the algorithms.

	Fumagalli *et. al.* [8]	Proposed Algorithms
Number of Token	Equal to the number of data-channels	One
Number of Transmitters and Receivers per node	Equal to the number data-channels	Two
Type of transmitter and receiver	All fixed	One tunable and one fixed
Fiber-Delay Lines	Exists at every node	None
Channel selection	Selects a free channel	EACP select earliest available channel and MSLP selects a channel with minimum scheduling latency
Transmission	A node can transmit and receive on more than one data-channels at the same time	A node can transmit on one data-channel and receive on another data-channel at the same time
Header processing	Header is processed at each intermediate node, and this introduces delay in the transmission	No processing of header takes place at intermediate nodes
Packet Removal	A packet is removed by the source that injected the packet into the channel	Packets are automatically removed at the destination node
Synchronization requirement	None	Yes

Table 9: Qualitative comparison of the proposed algorithms with Fumagalli *et. al.* [8]

The main disadvantage of Fumagalli *et al.* [8] is that the scheme is not scalable with the number of used wavelengths. With increase in the number of wavelengths, the number of transmitters and receivers has to be increased proportionately. Addition of new transmitter and receiver increases the running cost of the network. With multiple tokens

in the ring, the token maintenance is not an easy task. Such disadvantages of Fumagalli *et al.* [8] are the inherent advantages of our algorithm and are well addressed at the cost of increased delays. However, for error-free operation in our proposed algorithm all nodes are required to operate in synchronously in time-domain. This necessitates the use of a synchronous transport like SONET in the ring which will add to the cost of the network.

A qualitative comparison of the features of our proposed algorithms and Fumagalli *et. al.* algorithm [8] is summarized in Table 9.

6 Conclusions

In this paper, we proposed a node architecture and two algorithms which we called EACP and MSLP algorithms, respectively to access the shared medium and to provide QoS in optical ring networks. As desired in any MAC protocol, both the proposed algorithms are contention-free. The proposed node architecture uses a tunable transceiver and a fixed transceiver. The tunable transceiver is an emerging device and is in tune with the advances in laser technology, and makes the scheme scalable. We included performance of EACP and MSLP algorithms in terms of wavelength utilization, mean delay and blocking probability. We observed that the MSLP algorithm performs better in terms of channel utilization and delay because it selects the channel with minimal scheduling latency. Minimizing the channel scheduling latency increases the channel throughput and reduces the delay.

We compared our scheme with that of Fumagalli *et al.* [8]. We found that our scheme performed better in terms of wavelength utilization. However, the scheme proposed by Fumagalli *et al.* [8] has smaller delay and higher throughput in terms of data-rate in *bps*; this is mainly due to the difference in node architecture. The node architecture in our presented scheme has one tunable and fixed transceiver, while that of Fumagalli *et al.* [8] has an array of transmitters and receivers resulting in higher data-transfer. The proposed scheme is in tune with the advances in laser technology. It is widely believed that in future, nodes of WDM networks will be equipped with tunable transceiver(s) over an array of fixed transmitters and receivers at a little additional cost.

References

[1] R. Ramaswami and K. N. Sivarajan, *Optical Networks: A Practical Perspective*. Morgan Kaufmann, 1998.

[2] C. S. R. Murthy and G. Mohan, *WDM Optical Networks: Concepts, Design and Algorithms*. Prentice Hall of India Limited, 2002.

[3] K. Bengi and H. R. van As, "Efficient QoS Support in a Slotted Multihop WDM Metro Ring," *IEEE Journal on Selected Areas in Communications*, vol. 20, pp. 216 – 227, Jan. 2002.

[4] M. A. Marsan, A. Bianco, E. Leonardi, A. Morabito, and F. Neri, "All-Optical WDM Multi-Rings with Differentiated QoS," *IEEE Commun. Mag.*, pp. 58 – 66, Feb. 1999.

[5] M. A. Marsan, A. Bianco, E. Leonardi, F.Neri, and S.Toniolo, "An Almost Optimal MAC Protocol for All-Optical Multi-Rings with Tunable Transmitters and Fixed Receivers," 1997. http://citeseer.nj.nec.com/marsan97almost.html.

[6] M. A. Marsan, A. Bianco, and E. G. Abos, "All-Optical Slotted WDM Rings with Partially Tunable Transmitters and Receivers." http://citeseer.nj.nec.com/34986.html, 1996.

[7] M. J. Spencer and M. A. Summerfield, "WRAP: A Medium Access Control Protocol for Wavelength- Routed Passive Optical Networks," *Journal of Lightwave Technology*, vol. 18, pp. 1657 – 1676, Dec. 2000.

[8] A. Fumagalli, J. Cai, and I. Chlamtac, "A Token Based Protocol for Integrated Packet and Circuit Switching in WDM Rings," in *IEEE Globecom, Sedney*, November 1998.

[9] A. Fumagalli, J. Cai, and I. Chlamtac, "The Multi-Token Inter-Arrival Time (MTIT) Access Protocol for Supporting IP over WDM Ring Network," in *Proc. ICC'99 conf. Vancouver, Canada*, June 1999.

[10] J. Fransson, M. Johansson, M. Roughan, L. Andrew, and M. A. Summerfield, "Design of a Medium Access Control Protocol for a WDMA/TDMA Photonic Ring Network." http://citeseer.nj.nec.com/487875.html.

[11] A. K. Turuk, R. Kumar, and R. Badrinath, "A Token Based Distributed Algorithm for Medium Access in an Optical Ring Network," in *Proc. Int. Workshop Distributed Computing, Lecture Notes in Computer Science*, vol. 2918, pp. 340 – 349, December 2003.

[12] W. Cho and B. Mukherjee, "Design of MAC Protocol for DWADM-Based Metropolitan-Area Optical Ring Network," in *Global Telecommunications Conference, 2001, GLOBECOM'01, IEEE, Volume: 3, 25-29 Nov. 2001*, pp. 1575 – 1579, November 2001.

[13] D. A. Levine and I. F. Akyildiz, "PROTON: A Media Acess Control Protocol for Optical Network with Star Topology," *IEEE/ACM Transactions on Networking*, vol. 3, pp. 158 – 168, April 1995.

[14] F. Jia and B. Mukherjee, "The Receiver Collision Avoidance (RCA) Protocol For A Single-Hop WDM Ligthwave Network," *Journal of Lightwave Technology*, vol. 11, pp. 1053 – 1065, May/June 1993.

[15] B. Mukherjee, "WDM Based Local Light Wave Networks Part-i : Single Hop Systems," *IEEE Network*, pp. 12 – 27, May 1992.

[16] P. E. Green, L. A. Coldren, K. M. Johnson, J. G. Lewis, C. M. Miller, J. F. Morrison, R. Olshansky, R. Ramaswami, and E. H. Smith, "All Optical Packet-Switched Metropolitan-Area Network Proposal," *Journal of Lightwave Technology*, vol. 11, pp. 754 – 763, May/June 1993.

[17] L. G. Kazovsky and P. T. Poggiolini, "STARNET: A Multi-gigabit-per-second Optical LAN Utilizing a Passive WDM Star," *Journal of Lightwave Technology*, vol. 11, pp. 1009 – 1027, May/June 1993.

[18] O. K. Tonguz and K. A. Falcone, "Fiber-Optic Interconnection of Local Area Networks: Physical Limitation of Topologies," *Journal of Lightwave Technology*, vol. 11, pp. 1040 – 1052, May/June 1993.

[19] I. Chlamtac and A. Ganz, "Design Alternatives of Asynchoronous Star Networks," in *Proc. IEEE ICC'89*, vol. 2, pp. 739 – 744, June 1989.

[20] I. M. I. Habbab, M. Kavehrad, and C. E. W. Sundberg, "Protocol for Very High Speed Optical Fiber Local Area Networks Using Passive Star Topology," *Journal of Lightwave Technology*, vol. 5, pp. 1782 – 1794, December 1987.

[21] N. Mehravari, "Performance and Protocol Improvements for a Very High Speed Optical Fiber Local Area Networks Using a Passive Star Topology," *Journal of Lightwave Technology*, vol. 8, pp. 520 – 530, April 1990.

[22] G. N. M. Sudhakar, N. D. Georganas, and M. Kavehrad, "Slotted Aloha and Reservation Aloha Protocols for Very High Speed Optical Local Area Networks Using Passive Sstar Topology," *Journal of Lightwave Technology*, vol. 9, pp. 1411 – 1422, Oct. 1991.

[23] *Tunable Lasers to cut Optical Costs*. http://zdnet.com.com/2102-1103-986221.html.

[24] C.-K. Chan, L. kuan Chen, and K.-W. Cheung, "A Fast Channel-Tunable Optical Transmitter for Ultrahigh- Speed All-Optical Time-Division Multiaccess Networks," *IEEE Journal on Selected Areas in Communications*, vol. 14, pp. 1052 – 1056, June 1996.

[25] O. A. Lavrora, G. Rossi, and D. J. Blumenthal, "Rapid Tunable Transmitter with Large Number of ITU Channel Accessible in Less Than 5 ns," in *Proc. ECOC 2000 Vol. 2*, pp. 169 – 170, September 2000.

[26] B. Mason, G. A. Fish, S. P. Denbaars, and L. A. Coldren, "Widely Tunable Sampled Grating DBR Laser with Integrated Electroabsorption Modulator," *IEEE Photonic Technology Letters*, vol. 11, pp. 638 – 640, June 1999.

[27] C. R. Doerr, C. H. Joyner, and L. W. Stulz, "40-Wavelength Rapidly Digitally Tunable Laser," *IEEE Photonic Technology Letters*, vol. 11, pp. 1348 – 1350, Nov. 1999.

[28] A. Leon-Gracia and I. Widjaja, *Communication Networks: Fundamental Concepts and Key Architecture*. Tata McGraw-Hill Publishing Company Limited, 2000.

[29] O. K. Tonguz and K. A. Falcone, "Fiber-Optic Interconnection of Local Area Networks: Physical Limitation of Topologies," *Journal of Lightwave Technology*, vol. 11, pp. 1040 – 1052, May/June 1993.

[30] V. Paxson and S. Floyd, "Wide Area Traffic: The Failure of Poisson Modeling," *IEEE/ACM Transaction on Networking*, vol. 3, pp. 226 – 244, June 1995.

[31] T. D. Neame, M. Zukerman, and R. G. Addie, "A Practical Approach for Multimedia Traffic Modeling," in *Proceedings of Broadband Communication'99, Hong Kong, 10-12 November,1999*, pp. 73 – 82, 1999.