Published in Optics Communications,
2004, Vol 231, Iss 1-6, P 199-212

A token-based distributed algorithm for medium access in an optical ring network

A.K. Turuk *, R. Kumar, R. Badrinath

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

Abstract

In this paper, we propose a node architecture and a token-based algorithm to access the shared medium in an optical ring network. The algorithm is based on a reservation scheme. However, unlike other reservation schemes which operate in three stages viz. reserve, transmit and release, the proposed scheme operates in two stages viz. reserve and transmit and does not explicitly release the reserved resources. The proposed algorithm selects the earliest available data-channel for reservation hence we call it *earliest available channel* (EAC) algorithm. The *EAC* algorithm operates in a distributed manner, and has the capability of handling channel collision and destination conflicts. Each node in the network maintains three modules: *send* module, *receive* module and *token processing* module, and is equipped with a tunable transceiver in contrast to the previous work which uses an array of fixed transceivers. We study the performance of the algorithm, by simulation, for fixed-sized bursts and bursts whose size is determined by M/Pareto distribution. We find our algorithm is superior in terms of wavelength utilization. Since, our algorithm uses a single tunable transceiver at each node, this is scalable with respect to the number of data-channels, however, delays are marginally increased.

Keywords: Optical ring network; WDM; Medium access control; Token

1. Introduction

It is widely acknowledged that the rapid growth in demand for bandwidth due to the Internet explosion can be satisfied by optical networks, and in particular using the wavelength-division multiplexing (WDM) technology. A single fiber can support hundreds of wavelength channels. With the successful deployment of WDM in core networks, the access networks, e.g., local area networks (LANs) and metropolitan area networks (MANs) are bottlenecks. Recently, a lot of work has been reported in the literature for the deployment of WDM technology in the access network. In a LAN, the available bandwidth is shared among all the network users. To deal with multi-user access a media access control protocol is needed in such networks [1,2]. In recent years

many media access control protocols have been proposed for WDM LAN based on star, or a ring as the underlying physical topology.

There are three well-known access strategies for LANs based on ring topology viz. token ring, slotted ring and insertion register ring. These have been widely used as local area networks both in commercial systems and research prototypes. Ring network offers several attractive features such as higher channel utilization and bounded delay. WDM slotted rings are reported in [3–6]. Synchronization among the slots is a major design factor in slotted rings. Nodes must be synchronized so that a slot starts at the same time on all the wavelength channels in the network at the synchronization point. Synchronization points are the network hubs for star topology and the WDM ADMs for ring topology [7]. Bengi and van As [3] discussed multihop WDM metro ring where several nodes use the same channel for reception of packets. Marsan et al. [4] discussed an all-optical network where number of nodes in the network are equal to the number of wavelength channels. Marsan et al. [5] assigned separate slotted channels to a disjoint subset of destination nodes. A group of nodes share the same channel for reception. Fransson et al. [8] equipped the nodes with a WDM laser-array transmitter capable of transmitting at all wavelengths used in the network, and receive at a particular wavelength.

Token-based WDM ring networks are explained in [9,10]. Unlike *FDDI* rings, authors in [9,10] discussed multiple token in the ring. In [9,10], the number of tokens in the ring, the number of transmitter and receiver that each node is equipped with, is equal to the number of data-channels available in the ring which is one less than number of available wavelengths. In this paper, we propose a token-based algorithm called *earliest available channel* (EAC) algorithm to access the shared medium in a WDM ring network. The algorithm is based on a reservation scheme. An early version of this paper with a few initial results has been accepted for presentation in [11].

Unlike other reservation schemes that operate in three stages viz. reserve, transmit and release, the *EAC* algorithm operates in two stages viz. reserve and transmit. In our proposed scheme reserved resources are not explicitly released. 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. 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. The duration for which resources are reserved is different for different reservation requests. The reserved resources can be requested for reservation by another node after that period. This does not necessitate the explicit release of reserved resources. Two different nodes can make reservation requests for the same resource during the same cycle of the token but for different times. Transmitter of the source and receiver of the destination are tuned to the same reserved 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 as reported in [12–15] makes it possible to set up lightpath dynamically.

The *EAC* algorithm operates in a distributed manner. Each node in the network maintains three modules: *send* module, *receive* module and *token processing* module. A node invokes send module if its *req_made* queue (a queue that stores all the reservation request made by the node) is non-empty. Similarly, *receive* module is invoked if its *req_rec* queue (a queue that stores all the transmission request to the node) is non-empty. *Token processing* module is invoked when a node receives a token. *Req_made* queue and *req_rec* queue are updated by the *token processing* module. The algorithm has the capability of avoiding channel collision and destination conflicts. The reservation mechanism is explained in details in the subsequent sections.

We study the performance of the algorithm by simulation for different types of traffic including bursty traffic which we model using a M/Pareto distribution. We compare the performance with another token-based algorithm *Multi-Token Inter-Arrival Time* (MTIT) Access Protocol [10]. To the

best of our knowledge *MTIT* is the only token-based protocol proposed for optical ring networks. In the later part of the paper, we include a qualitative comparison of *MTIT* and *EAC* algorithms in Table 7.

The rest of the paper is organized as follows. In Section 2, we described the system model. In Section 3, the *EAC* algorithm is detailed. The proof of correctness of the algorithm is given in Section 4. Simulation results comparing *MTIT* and *EAC* algorithms are reported in Section 5. Finally, some conclusions are drawn in Section 6.

2. System model

2.1. Assumptions

There are N nodes in the network, numbered as $0, 1, 2, \ldots, N-1$. 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 $i, j \in 0, 1, \ldots, N-1$; node j is the successor of node i and node i is the predecessor of node j. R is the *Ring Latency*. The system supports W wavelengths $\lambda_0, \lambda_1, \ldots, \lambda_{W-1}$. There are $W-1$ *data-channels* and one *control-channel*. One of the wavelength, λ_0, is dedicated to a control-channel, and rest of the wavelengths are used as data-channels. A circuit is established on wavelength, λ_0, between every pair of adjacent nodes i and j. The circuit thus established is the dedicated control-channel. A pair of nodes i and j are said to be adjacent if j is the successor of node i and node i is the predecessor of node j.

A node architecture is shown in Fig. 1. Each node is equipped with a *fixed* and *tunable* transceiver. A similar architecture is proposed in [16]. However, the authors in [16] have equipped the nodes with a SONET ADM and a DWADM (dynamic wavelength add-drop multiplexers). The property of a DWADM dictates that, at any time, the input channel wavelength must be same as the output channel wavelength. A node equipped with DWADM, must transmit and receive on the same wavelength. This limits the usage of wavelengths and performance of the ring networks equipped with DWADM. To overcome the limitation of a node with DWADM, we equip the nodes with tunable transceiver, so that a node can transmit packets on one wavelength and receive packets on some other wavelength independently. The fixed transmitters and receivers are tuned to wavelength, λ_0, to transmit and receive *control information* between adjacent nodes. Tunable transmitters and tunable receivers are tuned to data-channels as and when required. For two nodes in the network to communicate, tunable transmitter of the source node and tunable receiver of the destination node must be tuned to the same wavelength (data-channel). Note that information transfer now takes a single hop over a data-channel. The system has a single *token* that circulates around the ring on the *control-channel*. The token consists of N fields which we call *slots* with *slot* i assigned to node i. Each field is subdivided into five mini-fields which we collectively called *control information* of a slot. We define a *Token Period* (TP) as the period between two successive receives of the token by a node. We calculate *TP* as $TP = R + N \times p$ where p is the processing delay of token at each node. Since *TP* is same for all nodes in the network, each node gets a fair chance to access the shared medium. Thus, the delay involved is bounded.

A node on receiving the token processes each slot, l $(0 \leqslant l < N)$ to update its knowledge about node, l, in the network. Prior to 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 slot in the token. Reservation mechanism is explained in Section 3.

Every node has $N-1$ buffers, one for every destination. Buffers handle packets in a FIFO order. The buffers are scanned by a pre-determined method to select packets to transfer.

2.2. Notations

Each node i maintains the vectors *DAT* and *CAT* where:

DAT[i]: Indicates the earliest time at which the transmitter of node i will be available for transmitting.

DAT[x]: Indicates the earliest time at which the receiver of node x will be available for receiving, where $x \neq i$.

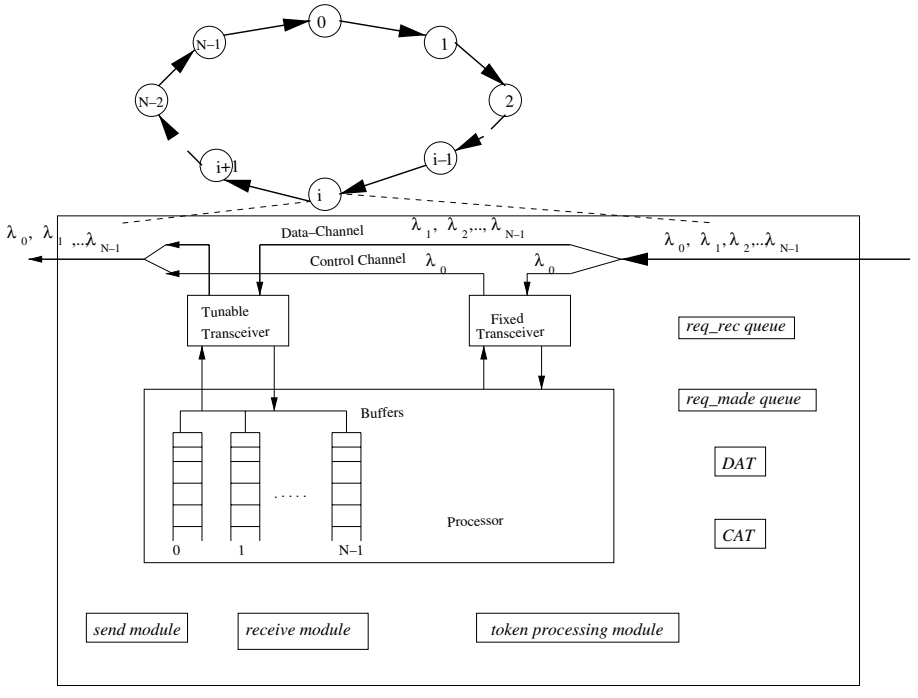

Fig. 1. Architecture of node i in the optical ring network.

$CAT[c]$: Indicates the earliest time at which the data-channel c will be available for transmission.

τ_t: Transmitter available time.

τ_d: Destination available time.

τ_c: Channel available time.

t_u: Tuning time of the transmitter/receiver.

t_p: Average propagation delay between source and destination, and

current_time: time at which an action is taken at a node.

req_made queue is a FIFO queue that holds the reservation requests made by a node. Each element of the queue has the following fields: *tuning_time* – time at which transmitter of a node is tuned to a data-channel, *destination_identity* – identity of destination node to which transmission will take place, *data_channel* – wavelength to which the transmitter of a node will be tuned to, *transmission_duration* – duration for which transmission will take place.

req_rec queue is 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 entered in its *req_rec queue*. No other nodes will make an entry of this request in its *req_rec queue*. Elements of the *req_rec queue* are same as that of *req_made queue*. Here the *data_channel* field specifies the wavelength to which the receiver of a node will be tuned to.

T_x: Indicates the status of a node's transmitter (*BUSY/FREE*).

R_x: Indicates the status of a node's receiver (*BUSY/FREE*).

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

s: 1 indicates node j is requesting for reservation; 0 indicates no request is made by node j.

d: Identity of the destination node requested for reservation.

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

t_c: Time at which receiver of the destination, and transmitter of the source are tune to data-channel c.
D: Duration of transmission.

3. EAC algorithm

In this section we describe a token-based distributed algorithm which we call *EAC* algorithm for medium access to a WDM ring networks. The inter-arrival time of token at each node remains the same (i.e., *TP*), giving each node a fair access to the shared data-channels. *EAC* algorithm guarantees that no collision destination and/or data-channel takes place and is based on a reservation scheme. However, it differs from the traditional reservation scheme in two aspects. First, no explicit release of reserved resources takes place. Second, unlike the traditional reservation scheme where resources are reserved only when they are free, *EAC* algorithm look ahead to find at what time the required resources are available, and reserve the resources from that point of time. Each node maintains status of its transmitter, receiver of other node's, and data-channels indicating the time at which they will be available for transmission.

Upon receiving the token, a node first updates its status vectors *DAT* and *CAT* maintained at its node. If its buffers are non-empty, then selects the burst with maximum waiting time and an earliest available data-channel. Node then makes the reservation request by writing the control information in its allotted slot of the token. Then the token is sent to its adjacent node. When a node receives back the token (i.e., after a period of *TP*), its reservation request is completed, and all the nodes have recorded the next availability of the requested resources in their status vector *DAT* and *CAT*. Before transmission, transmitter of the source and the receiver of the destination are tuned to the same data-channels in other words a circuit (lightpath) is established between the source and destination and remains established for the period of transmission which is determined at the time reservation request is made.

Algorithm has the following three modules: *Send* module, *Receive* module, and *Token Processing* module.

If a node's *req_made* queue is non-empty and the transmitter of the node is free, then the node invokes the *send* module. The front element of the *req_made* queue is removed and the status of the transmitter, T_x, is set to BUSY. Transmitter is tuned to the data-channel specified in the *data_channel* field of the element at the time specified in the *tuning_time* field of the 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 *transmission_duration* field of the element. After transmission is completed, the 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 eventhough the status of the transmitter is set to BUSY. Requests made by a node are added in the *req_made* queue in the order in which the requests are made. Note a request made by a node is added to the *req_made* queue only if the request is successful in making reservation.

Similarly, if a node's *req_rec* queue is non-empty and the receiver of the node is free, then the node invokes the *receive* module. The front element of the *req_rec* queue is removed and the status of the receiver, R_x, is set to BUSY. Receiver is tuned to the data-channel specified in the *data_channel* field of the element at the time specified in the *tuning_time* field of the element. Once the receiver of the node is tuned to the specified data-channel, node starts receiving information and continue for a duration specified in the *transmission_duration* field of the element. After the transmission is completed, the status of the receiver, R_x, is set to FREE, and another transmission process begins. Note the receiver of the node is tuned to the specified data-channel only at the specified time eventhough the status of the receiver is set to BUSY.

When a node receives the token it invokes the *token processing* module. Following actions are taken by the *token processing* module. First, it updates the *DAT* and *CAT* vectors maintained at

the node. If the buffers are non-empty, then it finds the destination identity of the burst with maximum waiting time, and an earliest available data-channel. The *maximum* of the time at which the node's transmitter, destination node's receiver and the selected data-channel is free, is found. Let this time be t' and this gives the time at which the required resources viz. 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'. Suppose the node has received the token at t. Its reservation process is completed at $t + TP$. If $t' < t + TP$, this 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 $t + TP$. Hence, 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 its 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 the reservation requested is inserted in its *req_send* queue.

For transmission to take place between source and destination, the transmitter of the source and the receiver of the destination must be tuned to the same data-channel at the same time. We have shown later that the transmitter of the source and the receiver of the destination are tuned to the same data-channel precisely at the same time.

3.1. Pseudocode of EAC algorithm

Perform the following *Cases* at each node i:
CASE: **if** *req_made* queue is non-empty and $T_x = FREE$ **then** invoke the *Send* module
CASE: **if** *req_rec* queue is non-empty and $R_x = FREE$ **then** invoke the *Receive* module
CASE: Invoke the *Token processing* module when a *node i* receive the token

3.1.1. Send module

1. Remove the front element of the *req_made* queue. Let it be $req\_made(l)$
2. Set $T_x = BUSY$
3. ***if*** $current\_time \geqslant req\_made(l) \cdot tuning\_time$ ***then***
   - Tune the transmitter to data-channel, $req\_made(l) \cdot data\_channel$
4. ***if*** $current\_time \geqslant req\_made(l) \cdot tuning\_time + t_u$ ***then***
   - Transmit data to node, $req\_made(l) \cdot destination\_identity$
5. ***if*** $current\_time \geqslant req\_made(l) \cdot tuning\_time + t_u + t_p + req\_made(l) \cdot transmission\_duration$ ***then***
   - Set $T_x = FREE$

3.1.2. Receive module

1. Remove the front element from the *req_rec* queue. Let it be $req\_rec(l)$
2. Set $R_x = BUSY$
3. ***if*** $current\_time \geqslant req\_rec(l) \cdot tuning\_time$ ***then***
   - Tune the receiver to data-channel, $req\_rec(l) \cdot data\_channel$
4. ***if*** $current\_time \geqslant req\_rec(l) \cdot tuning\_time + t_u$ ***then***
   - Receive data
5. ***if*** $current\_time \geqslant req\_rec(l) \cdot tuning\_time + t_u + t_p + req\_rec(l) \cdot transmission\_duration$ ***then***
   - Set $R_x = FREE$

3.1.3. Token processing module

1. Examine $slot_i(s, d, c, t_c, D)$ of the token *if* $(s = 1)$ then do the following:
   - Set s field of $slot_i$ to *zero*
   - Add the request in *req_send* queue of node i
   - Set $DAT[d] = DAT[i] = CAT[c] = t_c + t_u + t_p + D$ /* request made by node i is added to the *req_send* queue of node i. Availability of destination node d's receiver, transmitter of node i, and data-channel c are updated in DAT and CAT of node i */
2. For all $slot_j(s, d, c, t_c, D)$ of the token, $j \neq i$ do the following:
   - *if* ($s = 1$ and $t_c + t_u + t_p + D > DAT[d]$) *then*
     - $DAT[d] = t_u + t_c + t_p + D$
   - *if* ($s = 1$ and $t_c + t_u + t_p + D > CAT[c]$) *then*
     - $CAT[c] = t_u + t_c + t_p + D$

   /* node i updates its knowledge about the availability of receiver of other nodes, and data-channel by appropriately updating its DAT and CAT vector */
3. *if* node i's buffers are empty goto step 13
4. Find a burst with maximum waiting time. Let the destination identity of the burst be say x

5. Find the earliest available channel $k = \{m : CAT[m]$ is minimum for $m = 1, \ldots, W-1\}$
6. $\tau_t = DAT[i]$
7. $\tau_d = DAT[x]$
8. $\tau_c = CAT[k]$
9. Find $\tau = \max(\tau_t, \tau_d, \tau_c)$. This gives the earliest time at which the transmitter of node i, receiver of destination node x and the data-channel k are available at the same time, and can be reserved by node i
10. ***if*** $\tau <$ *current_time* + *TP* ***then*** $\tau =$ *current_time*+ *TP* /* resource reservation time is updated to reservation completion time if the required resources are available before the reservation is completed */
11. Compute the duration of transmission, D
12. Write the control information in $slot_i(s = 1, d = x, c = k, t_c = \tau, D)$ of the token
13. Send the token to successor node of i

3.1.4. Example

In the following example, we consider a four node ring-network to illustrate the reservation process. The number of data-channel is assumed to be 2. Let the contents of *DAT* and *CAT* vectors at time t be as shown in Table 1. Table 2 shows the traffic matrix at time t. The entry $b(x, y)$ corresponding to row m and column n of Table 2 indicates, node m has a burst destined to node 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. We assume the following value for the parameters: $TP = 20, t_u = 2$, propagation delay of token between adjacent node be 5 and the processing delay of token at each node is assumed to be negligible. Computed value of t_p as given in [17] is 10.

Table 1
Contents of DAT and CAT vectors at the nodes at time t

DAT[0] = 0, DAT[1] = 5, DAT[2] = 7, DAT[3] = 10
CAT[λ_1] = 5, CAT[λ_2] = 7

Table 2
Traffic matrix at time t

Node	0	1	2	3
0		$b(10, 4)$	$b(4, 3)$	
1	$b(20, 4)$		$b(25, 3)$	
2		$b(10, 3)$		
3	$b(10, 5)$			

Let $t = 6$ and node 0 received the token at t. Node 0 selects the burst with maximum waiting time i.e, burst destined to node 2, and a earliest available data-channel i.e., λ_1. The following computation is performed: $\tau_t = 0(DAT[0])$, $\tau_d = 7(DAT[2])$, $\tau_c = 5(CAT[\lambda_1])$, $\tau = 7(max(\tau_t, \tau_d, \tau_c))$. $\tau < t + TP$ so the value of τ is set to $t + TP$, i.e., 26. Node 0 writes control information in $slot_0(s = 1, d = 2, c = \lambda_1, t_c = \tau, D = 4)$ of the token and sends it to its successor, node 1. Node 1 on receiving the token updates the contents of *DAT* and *CAT* vectors shown in Table 3.

Node 1 selects the burst destined to node 2 and the earliest available data-channel λ_2. Following computation is performed: $\tau_t = 5$, $\tau_d = 47, \tau_c = 7$, $\tau = 47$. When node 1 receives the token, the value of t is updated to $t + 5$ (i.e., $t +$ the propagation delay between adjacent nodes which we have assumed to be 5 in our example). Node 1 writes control information in $slot_1(s = 1, d = 2, c = \lambda_2, t_c = \tau, D = 25)$ and sends it to node 2.

Updated values of *DAT* and *CAT* vectors at node 2 are shown in Table 4. Request from node 0 and node 1 are entered in the *req_rec* queue of node 2.

Node 2 selects the burst destined to node 1 and the data-channel λ_1. Following computation is performed: $\tau_t = 7, \tau_d = 5, \tau_c = 47, \tau = 47$. Node 2 writes control information in $slot_2(s = 1, d = 1, c = \lambda_1, t_c = \tau, D = 10)$, and then sends it to node 3.

Updated values of *DAT* and *CAT* vectors at node 3 are shown if Table 5.

Node 3 selects the burst destined to node 0 and data-channel λ_1. Following computations is performed: $\tau_t = 10, \tau_d = 0, \tau_c = 69, \tau = 69$. Node 3

Table 3
Contents of DAT and CAT vectors at node 1

DAT[0] = 0, DAT[1] = 5, DAT[2] = 47, DAT[3] = 10
CAT[λ_1] = 47, CAT[λ_2] = 7

Table 4
Contents of DAT and CAT vectors at node 2

DAT[0] = 0, DAT[1] = 5, DAT[2] = 7, DAT[3] = 10
CAT[λ_1] = 47, CAT[λ_2] = 84

Table 5
Contents of DAT and CAT vectors at node 3

DAT[0] = 0, DAT[1] = 69, DAT[2] = 84, DAT[3] = 10
CAT[λ_1] = 69, CAT[λ_2] = 84

Table 6
Contents of DAT and CAT vectors at node 0

DAT[0] = 47, DAT[1] = 69, DAT[2] = 84, DAT[3] = 10
CAT[λ_1] = 91, CAT[λ_2] = 84

writes control information in $slot_3(s = 1, d = 0, c = \lambda_1, t_c = \tau, D = 10)$ and then sends the token to the Node 0.

When node 0 receives the token, it adds the reservation request in its *req_made* queue. The contents of *DAT* and *CAT* vectors after processing the token are shown in Table 6.

The above example illustrate how the reservation is made and the values of *DAT* and *CAT* vectors updated at each node of the ring for given traffic matrix.

4. Correctness of the algorithm

4.1. Destination collision never occurs

Destination collision occurs when a node receives data from more than one nodes at the same time. In the following section, we prove that, in our proposed algorithm, destination collision never occurs.

Suppose destination collision occurs at node z. This implies that, node z has received data from more than one nodes say, node x and node y at same time. We show that node y (node x) can transmit to node z only after the completion of transmission from node x (node y).

We know that source must reserve the destination before it transmits. Suppose node x receives the token at t, and makes reservation request for node z. Let the value of τ (the time at which node x reserves the destination node z) in step 10 of token processing module be T, and the transmission duration be D_x. In other words, we can say, node x completes its transmission to node z at $T + t_u + t_p + D_x$.

Suppose node y receives the token at $t' > t$ such that $t < t' < t + TP$, and makes reservation request for node z. We show that node y makes reservation request to node z only after node x's transmission to node z is completed. Node y first updates the value of $DAT[z]$ setting $DAT[z] = T + t_u + t_p + D_x$ when it receives the token. To make reservation request to node z, the time at which receiver of node z is available, is calculated. This is given by $\tau_d = DAT[z]$ in step 7 of the *token processing* module. The value of τ in step 10 of the *token processing* module gives the earliest time at which node y can reserve node z.

The earliest time at which node y can reserve node z is $\tau \geqslant DAT[z] = T + t_u + t_p + D_x$. But node x completes its transmission to node z at $T + t_u + t_p + D_x$. This implies node y can reserve node z only after node x completes its transmission. Which further implies that node y transmits only after node x completes it's transmission. Thus, destination collision does not occur.

4.2. Channel collision never occurs

Channel collision occurs only when more than one nodes transmit on the same data-channel.

Similar to Section 4.1 we can show that channel collision never occurs.

4.3. Reservation request by a node does not overlap in time

Reservation request made by a node overlaps in time only when a new transmission is initiated before the completion of the on-going transmission.

Let a node x make two reservation requests say, destined to nodes y and z, respectively, that overlap in time. In other words transmission to node z (node y) initiated before the completion of transmission to node y (node z). Here we show that the transmission to node z (node y) is initiated after the completion of transmission to node y (node z).

We know that when a node receives the token, it makes a single reservation request. Let node x receive the token at t, and make reservation request to node y. Let T be the time at which node x reserves the resources, and duration of transmis-

sion be D_y. This implies transmitter of node x will be free at $T + t_u + t_p + D_y$.

When node x receives back the token it updates its transmitter's available time $DAT[i] = T + t_u + t_p + D_y$. Let node x make another request to node z. The transmitter's available time $\tau_t = DAT[i] = T + t_u + t_p + D_y$ is found in step 6 of the *token processing* module. Step 10 of the *token processing* module gives the earliest time at which node x can reserve the resources corresponding to its request to node z.

The earliest time at which resources are reserved by node x corresponding to its request to node z is $\tau \geqslant DAT[i] = T + t_u + t_p + D_y$. This implies that node x's transmission to node z is initiated after completion of its transmission to node y.

Hence, the requests from node x to nodes y and z, respectively, do not overlap in time. This contradicts our assumption. Thus, the reservation request made by a node does not overlap in time.

4.4. Reservation request received by a node does not overlap in time

Similar to Section 4.3 it can be shown that reservation request received by a node does not overlap in time.

4.5. Transmitter of the source and the receiver of the destination are tuned to the same data-channel precisely at the same time

For transmission to take place, transmitter of the source and receiver of the destination must be tuned to the same data-channel at the same time.

We know that when a request is in *req_made* queue of a node, a copy of that request will also be there in the *req_rec* queue of another node. Further, *send* and *receive* modules process the requests in the order in which the requests exist in the *req_made* and *req_rec* queues of the node. So when the transmitter of a node i is tuned to data-channel c at time t, there exists another node j whose receiver is also tuned to data-channel c at time t.

Let node x make reservation request to node z. Let the data-channel selected be λ and the time at which node x reserves the resources be T.

When node x makes reservation request to node z: (1) step 6 of the token processing module guarantees that all the previous transmission from node x is completed at τ_t, (2) step 7 of the token processing module guarantees that all previous transmissions to node z are completed at τ_d, (3) step 8 of the token processing module guarantees that previous transmission on data-channel λ is completed at τ_c.

Step 10 of the token processing module gives the earliest time at which transmitter of node x and receiver of node z are tuned to the data-channel λ which we assume to be $T \geqslant max(\tau_t, \tau_d, \tau_c)$. This implies at T: (1) All transmissions corresponding to previous requests made by node x are completed, (2) All transmissions corresponding to previous requests to node z are completed, (3) No transmission is taking place on data-channel λ.

Let *send* module of node x remove the request to node z from the *req_made* queue of node x and tune the transmitter of node x data-channel λ at T. This also implies all requests made to node z before node x's request, have completed their transmissions, and *receive* module of node z has removed the request of node x from *req_rec* queue of node z and tuned the receiver of node z at T to data-channel λ.

Similarly, let the *receive* module of node z remove the request of node x from the *req_rec* queue of node z and tune the receiver of node z to data-channel λ at T. This also implies node x has completed transmission of all requests that it has made before its request to node z, and the *send* module of node x has removed the request to node z from *req_made* queue of node x and tuned the transmitter of node x at T to data-channel λ.

Thus we conclude that the transmitter of the source and receiver of the destination are tuned to same data-channel precisely at the same time.

5. Simulation results

We evaluate the performance of the EAC algorithm by simulation. We measure the performance in terms of throughput and mean packet delay. Throughput is defined as through-

put = (transmission time)/(transmission time + scheduling latency), and is a measure of how efficiently data-channels are utilized in the network. We consider a 10-node ring network. Nodes are spaced equally in the ring. The number of data-channels is taken to be five as a default value; for others, it is mentioned along with the results. We assume the following values for other parameters: length of the ring is 100 kms, processing time of token at each node is 1 μs, tuning time of transmitter and receiver is 5 μs. Computed value of TP is 510 μs. The average propagation delay, t_p, between nodes is computed as given in [17], $t_p = N/2 \times \tau$ where τ is the propagation delay between adjacent nodes.

We use different types of data traffic – fixed-sized bursts and bursts whose size is determined by M/Pareto distribution. We consider bursty traffic as the traffic in LANs is reported to be bursty in nature [18]. We use M/Pareto distribution for generating the bursts of varying sizes [19]. We keep the size of packets in a burst to be fixed at 10 kb per packet. We have taken capacity of the data-channel to be 1 Gbps. A node can transmit at a maximal speed of 1 Gbps. Bursts of fixed size and fixed inter-arrival time do not generate bursts of size exceeding this capacity. However, the bursts generated from M/Pareto distribution which follow an exponential distribution for inter-arrival time may generate bursts exceeding this capacity. In the following subsections, we have included results of the traffic generated from the following three types of cases :

- Case 1: Both size and inter-arrival 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 use the following notations in terms of *DaDb*, *EaDb* and *EaPb*:

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

The *D*, *P* and *E* are suffixed by *a* and *b*, which we use for denoting inter-arrival and burst size respectively. For example, *EaPb* implies exponential inter-arrival of bursts and Pareto distributed size of the bursts. For deterministic cases, the size of the burst and the inter-arrival of the burst are known a priori. We have included results for deterministic cases of data (*DaDb*), mainly, for understanding differences in MTIT and EAC algorithms. However, in the real-world scenario, neither the burst-size nor the inter-arrival of bursts are known a priori. Therefore, the real-world scenario is better modeled by *EaPb* cases.

5.1. Burst size vs. throughput

First, we include the plots for burst size vs. throughput in – Figs. 2–4. Burst size is expressed in number of packets and throughput is a measure of wavelength utilization in the network.

Throughput for fixed arrival and fixed-sized bursts is plotted in Fig. 2. The inter-arrival of bursts is assumed to be 1 ms. From the plots, it is observed that the throughput in both EAC and MTIT algorithms increases with increase in the burst size; this is mainly due to the increase in 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. Throughput increases linearly with burst size for both EAC and MTIT up to a certain limit, and then this gets

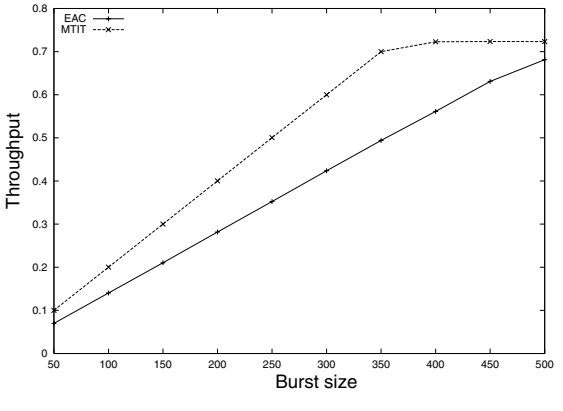

Fig. 2. Burst size vs. throughput for *DaDb*.

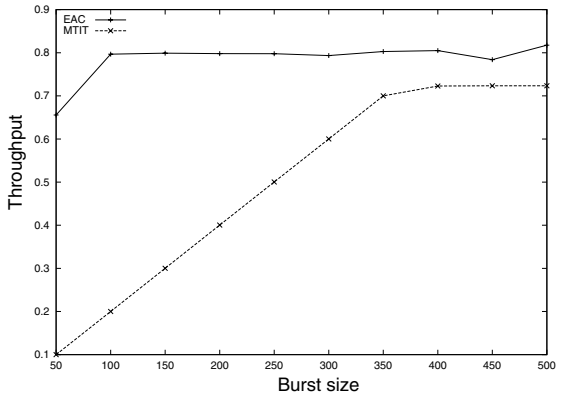

Fig. 3. Burst size vs. throughput for *EaDb*.

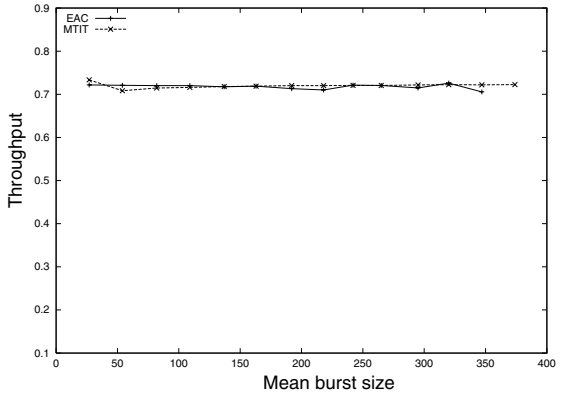

Fig. 4. Mean burst size vs. throughput for *EaPb*.

saturated for MTIT comparatively at a lower range. However, it continues to increase up to a larger burst-size (the range is not shown in the Fig. 2) for EAC. In MTIT, the duration of transmission by a node is the token holding time which is determined when a node captures the token, and there are multiple tokens.

For larger bursts, there are fewer nodes to transmit for an equal amount of data generated. In case of MTIT, a node has to transmit for a fixed duration, and to make more than one attempt to complete the transmission. In contrast, in EAC algorithm, a node will finish the complete transmission of higher bursts in a single attempt, and thus reduces the scheduling latency. Second, a wavelength remains unutilized till another node has data to transmit, in MTIT. However, this idle period of wavelength is lesser, for EAC, because it reserves immediately after its release. Thus, the total latency is smaller for EAC over MTIT algorithm. This difference is more significant for larger bursts. This behavior is reflected in the plots included in Fig. 2.

Next, we include results of fixed-sized bursts with an exponential inter-arrival time in Fig. 3. It is observed again that the throughput increases linearly for MTIT with increase in burst size and at a higher burst size it gets saturated; the reasons are stated above. However, in case of EAC algorithm, the throughput remains almost constant. There are two contributing factors to this phenomenon which governs the scheduling latencies and the idle period of wavelength utilization. One is the availability of the wavelength, as cited in the previous case, and second is the availability of destination node-receiver. Thus, for exponential distribution of inter-arrival time, we get an almost uniform wavelength utilization factor. However, this is not the case with MTIT, because, they have multiple receivers.

Finally, we include results of bursts taken from a Pareto distribution with an exponential inter-arrival time in Fig. 4. In this case, throughput remains almost constant throughout the burst-sizes for both the algorithms. Pareto distribution may generate a few bursts of very large size, and thus we have collected the simulation results for *mean* burst-size. We attribute contribution to the throughput response to two factors. First is from the varying burst size (Fig. 2), and the second is due to the inter-arrival burst-time (Fig. 3). An aggregation over burst-sizes and a fusion of this inter-arrival time are giving us an almost constant wavelength utilization across the mean burst-size.

This is a very interesting phenomenon and needs more investigation. Most of the work in literature has been confined to the data generated from Poisson distribution. It is almost established now that the real-data, in particular multimedia data, is self-similar and better modeled by a Pareto distribution. Not much analytical results are available for such self-similar data. An analytical study of the M/Pareto distribution along with the analysis of such algorithms may give deeper insight. This will also verify the simulation results.

This is an active area of research to be investigated further.

5.2. Burst size vs. mean delay

Next, we simulate results for burst size vs. mean delay and include in Figs. 5–7. Mean delay is an end-to-end delay and is the sum of access delay, propagation delay and the transmission time. Since, we have fewer data-channels than the number of nodes, the major contributing factor to mean-delay is the access delay. Access delay depends on scheduling latency on the channel. For EAC, scheduling latency is the time-duration between start of transmission and reserving a channel. For MTIT, it is duration between start of current transmission and completion of the previous transmission.

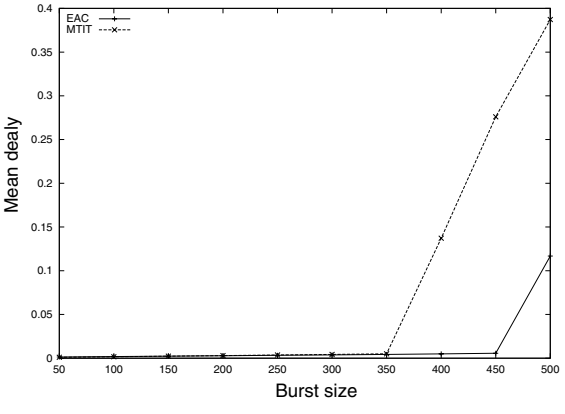

Fig. 5. Burst size vs. mean delay for *DaDb*.

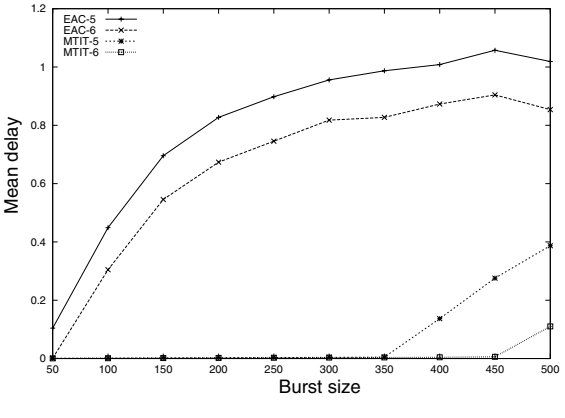

Fig. 6. Burst size vs. mean delay for *EaDb* using five and six data-channels.

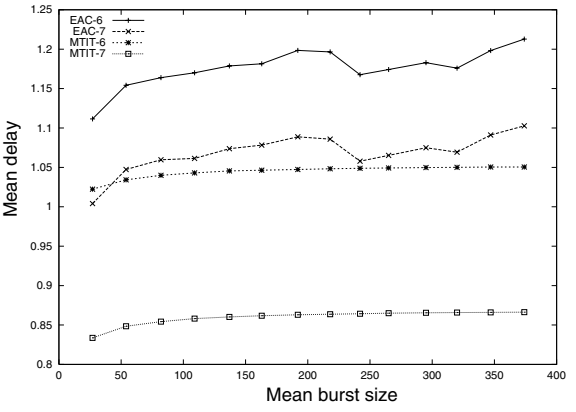

Fig. 7. Mean burst size vs. mean delay for *EaPb* using six and seven data-channels.

When both inter-arrival time and burst-size are deterministic, delays for smaller burst-sizes are insignificant for both EAC and MTIT as shown in Fig. 5. However, at larger burst-size, MTIT experiences higher delays; this is because of larger scheduling latency that MTIT experiences for larger bursts resulting in slower access to data-channel.

For exponential inter-arrival time and deterministic burst size, delays are plotted in Fig. 6. It is observed that mean delay experienced in EAC increases with burst size; this is due to the proportionate increase in the scheduling latency resulting in slower access and large delay. For MTIT, at higher burst size the scheduling latency increases giving higher delay. It is also observed in Fig. 6 that with increase in the number of data-channels the delay experienced by both EAC and MTIT decreases. This is in conformity with WDM technology.

The above observation remains the same for the case of *EaPb* shown in Fig. 7. In both the cases of *EaDb* and *EaPb*, delays are smaller for MTIT over EAC algorithm. Since, MTIT uses a fixed array of transmitters and receivers whose number is equal to the number of data-channels that each node can have, delays are smaller in comparison to EAC node which has a single transceiver. In MTIT, a node can transmit to more that one node concur-

Table 7
Comparison of MTIT with EAC algorithms

	MTIT	EAC
Number of tokens	Equal to the number of data-channels	One
Number of transmitters and receivers per node	Equal to the number of data-channels	Two
Fiber-delay lines	Exist at every node	No
Channel selection strategy	A channel is selected on capturing a token	Selects the earliest available channel
Transmission	Simultaneous transmission on each data-channel is possible at a node	A single transmission on a data-channel takes place at a node
Header processing	Header is processed at each intermediate node	No processing of header takes place
Packet removal	Removed by the source	Removed at the destination
Token arrival	Inter-arrival of token at a node may differ	Inter-arrival of token at each node remains same

rently resulting in smaller delays. Obviously, with increase in the number of data-channels, delay decreases in both the cases (Figs. 6 and 7).

Fixed array of transmitters and receivers in MTIT has some disadvantages too. For example, for a change in wavelength requirement, number of transmitters and receivers has to be changed accordingly at each node. Second, if the spectrum requirement changes the old (fixed) transceivers have to be replaced by new ones. Moreover, it is difficult to support QoS in MTIT [10] which is essential for the next generation optical networks. This is again an area of active research.

The qualitative comparison between *MTIT* and *EAC* is given in Table 7.

6. Conclusions

We proposed a token-based distributed algorithm *EAC* for medium access in a optical ring network. *EAC* is based on a reservation scheme, and has the capability of avoiding channel collision and destination conflicts. The scheme looks-ahead to find the time at which the required resources are available using a simple data structure and reserves them from the point of time of their release. We compared *EAC* algorithm with MTIT algorithm. We used a tunable transceiver to overcome the scalability problem associated with MTIT; this approach is in tune with the current advances in laser technology. From simulation we found that, the performance of *EAC* is superior in terms of throughput and delay for the traffic whose characteristics are known a priori. However, the performance is almost comparable for the bursts following a M/Pareto distribution for their size and an exponential distribution for the inter-arrival time. This performance of EAC is achieved using a single tunable transceiver rather than a fixed array of components. MTIT also pays the penalty in terms of scalability of data-channels, cost and maintenance. If the spectrum requirement changes the fixed lasers of MTIT have to be replaced with new ones, which is not the case in *EAC* algorithm. Since the number of transmitters at a node is equal to the number of data-channels, in MTIT any change in the data-channel requirement necessitates the change in the number of transmitters and receivers at each node. This increases the cost of the network. With multiple tokens equal to the number of data-channels in the network, the maintenance of token in the ring is not too trivial. These are the inherent advantages of our proposed *EAC* algorithm. Currently we are working to extend this work to include QoS provisioning.

References

[1] R. Ramaswami, K.N. Sivarajan, Optical Networks: A Practical Perspective, Morgan Kaufmann, Los Altos, CA, 1998.

[2] C.S.R. Murthy, G. Mohan, WDM Optical Networks: Concepts, Design and Algorithms, Prentice-Hall of India Ltd., 2000.

[3] K. Bengi, H.R. Van As, IEEE Journal of Selected Area in Communication 20 (January) (2002) 216.

[4] M.A. Marsan, A. Bianco, E. Leonardi, A. Morabito, F. Neri, IEEE Communications Magazine (Feb.) (1999) 58.
[5] M.A. Marsan, A. Bianco, F.E. Leonardi, S. Toniolo, An almost optimal MAC protocol for all-optical multi-rings with tunable transmitters and fixed receivers, 1997. Available from: <http://citeseer.nj.nec.com/marsan97almost.html>.
[6] M.A. Marsan, A. Bianco, E.G. Abos, All-optical slotted WDM rings with partially tunable transmitters and receivers, 1996. Available from: <http://citeseer.nj.nec.com/34986.html>.
[7] M.J. Spencer, M.A. Summerfield, Journal of Lightwave Technology 18 (December) (2000) 1657.
[8] J. Fransson, M. Johansson, M. Roughan, L. Andrew, M.A. Summerfield, Design of a medium access control protocol for a WDMA/TDMA photonic ring network. Available from: <http://citeseer.nj.nec.com/487875.html>.
[9] A. Fumagalli, J. Cai, I. Chlamtac, in: Proc. IEEE GLOBECOM, Sedney, November 1998.
[10] A. Fumagalli, J. Cai, I. Chlamtac, in: Proc. ICC'99, Vancouver, Canada, June 1999.
[11] A.K. Turuk, R. Kumar, R. Badrinath, in: Proc. Int. Workshop on Distributed Computing, Kolkata, India, 2003.
[12] C.-K. Chan, L. kuan Chen, K.-W. Cheung, IEEE Journal of Selected Areas in Communication 14 (June) (1996) 1052.
[13] O.A. Lavrora, G. Rossi, D.J. Blumenthal, in: Proc. ECOC 2000, vol. 2, September 2000, pp. 169–170.
[14] B. Mason, G.A. Fish, S.P. Denbaars, L.A. Coldren, IEEE Photon Technology Letters 11 (June) (1999) 638.
[15] C.R. Doerr, C.H. Joyner, L.W. Stulz, IEEE Photon Technology Letters 11 (Nov.) (1999) 1348.
[16] W. Cho, B. Mukherjee, Design of MAC protocol for DWADM-based metropolitan-area optical ring network, in: Proc. IEEE GLOBECOM, vol. 3, November 25–29, 2001, pp. 1575–1579.
[17] O.K Tonguz, K.A. Falcone, Journal of Lightwave Technology 11 (May/June) (1993) 1040.
[18] V. Paxson, S. Floyd, IEEE/ACM Transactions on Networking 3 (June) (1995) 226.
[19] T.D. Neame, M. Zukerman, R.G. Addie, in: Proc. Broadband Communication Conf., Hong Kong, 1999, pp. 73–82.